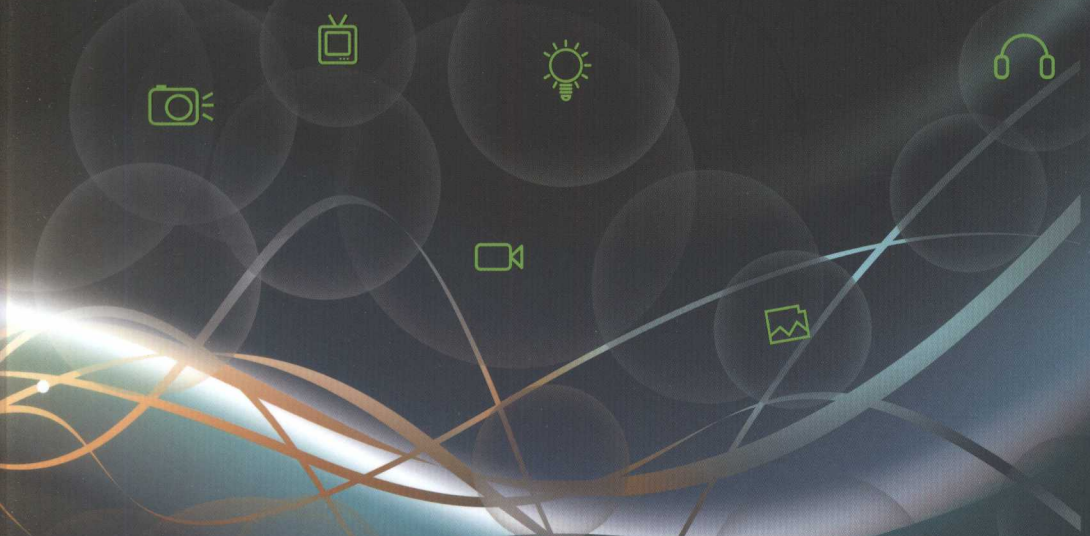
从零到一打造亿级用户应用的捷径
从小到大快速连接市场成为独角兽

微信小程序开发快速入门

黄曦 / 沙拉依丁·苏里坦 著



中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

黄曦

轻课 CTO，美国加州大学计算机科学系硕士，研究方向偏向于后端高负载，高可用架构。

曾就职于美国硅谷某著名互联网金融公司，担任全栈工程师，对 scala、clojure 等函数式语言有深入研究。

后作为技术合伙人加入轻课回国创业，从零到一打造基于微信公众号体系的百万级用户产品。

现在专注于企业技术管理，产品创新以及研发协作流程优化。

沙拉依丁 · 苏里坦

从高二开始自学编程，高三开发出首款应用程序并上线。工作后从 Java 开始全面踏入网站全栈开发领域，有着丰富的 Java、C#、PHP 全栈开发经验。随后根据自己的兴趣逐渐转做前端开发。于 2015 年入职乐视，从事前端工程师一职，期间曾带领团队完成乐嗨直播移动前端的开发，并完成多个乐视网前端业务的开发与优化，拥有 4 年以上前端的开发经验。后于 2016 年中入职轻课担任前端架构师职务，负责轻课前端架构的优化与新业务的拓展开发。由于对新技术的热衷，从最初得到消息即一直保持着对微信小程序的关注，在轻课团队有幸获得首批小程序内测资格后，根据公司需求完成了多个微信小程序 demo。对前端技术与小程序的开发有深入的研究。目前专注于前端技术架构的研究与实施，优化前端用户体验，加强技术团队间的协作。

微信小程序
开发快速入门

黄曦 / 沙拉依丁·苏里坦 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书全面系统地讲解了微信小程序入门知识。开篇讲解了小程序的特点与开发逻辑，以及如何申请和创建一个小程序项目与环境搭建，接着通过多个小程序实例来全面体验和讲解开发框架、实现过程及主要代码框架等，然后介绍小程序组件、开发方式、网络、缓存、位置和界面交互，以及开发过程与组件的应用技巧、各个微信原生 API 接口小程序开发的技巧等。每章具有多个小程序实战案例，让读者快速掌握该章所讲的知识，并实践小程序各项功能的应用及使用技巧。

本书结构清晰，由浅入深，可帮助读者快速掌握小程序的开发。适合于各种前端开发者，以及各种 APP 设计、开发和自学者。

图书在版编目（CIP）数据

微信小程序开发快速入门 / 黄曦，沙拉依丁·苏里坦著. —北京：电子工业出版社，2017.6

ISBN 978-7-121-31331-8

Ⅰ. ①微… Ⅱ. ①黄… ②沙… Ⅲ. ①移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2017）第 076300 号

策划编辑：刘　伟
责任编辑：刘　伟
印　　刷：北京季蜂印刷有限公司
装　　订：北京季蜂印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：720×1000　1/16　印张：18.5　字数：355 千字
版　　次：2017 年 6 月第 1 版
印　　次：2017 年 6 月第 1 次印刷
定　　价：59.80 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前　言

为什么写本书

2017 年 1 月 9 日，小程序如期发布，给本就异常火爆的前端领域又注入了一股新的力量，甚至很多公司已经开始招聘小程序开发者，其应用也呈现出蓬勃发展的趋势。

然而开发人员中也出现了一些不同的声音，有人认为它的出现又为前端领域增加了负担，因为与现有的 Web 标准不同，它需要前端开发者开发了应用后，再去适配另一个小程序环境，以此来批评微信不拥抱标准，自立门户，不够开放。

实际上在我们看来，小程序并没有为前端人员增加负担，反而为前端开发者创造了新的价值。因为小程序是更接近原生 APP 的一个新的开发框架，不符合 HTML5 标准，在这个意义上，它实际上是为开发者提供了新的开发渠道，虽然小程序并未提供类似 APP 的应用商城为开发者带来应用分发的经济效益，但小程序本身内置提供支付功能的 API，便于更快更方便地开发出既为用户带来价值又能为开发者或企业带来收益的小程序应用，这本身对开发者和企业而言，有足够的想象和拓展空间。

另外，小程序虽然不符合现有的 HTML5 标准，但是依然沿用了 JavaScript、CSS 以及 HTML 的语法基础，其 JavaScript 甚至支持 ES6 语法，并且小程序在这些基础上做了不少的扩充，并引入了新的 MINA 框架，开发者需要采用该框架进行开发，但其上手难度对于较为熟练的前端开发人员来说，门槛非常低。

而与“增加了前端人员负担”的观点相反，无论是在安卓、iOS 平台，还是平台下不同屏幕大小的手机，微信都通过小程序框架，为小程序提供了一致的呈现效果，在这一点上，反而彻底将前端开发人员从机型适配的苦海中解脱了出来。

而且对于小程序开发，微信提供了完整的开发编译环境，在这个层面上，也为前端人员节省了大量的环境架构工作量，使用微信开发者工具，可以立即着手开发小程序。

目前常用的小程序已超 150 个，基本涵盖了生活中的大部分场景，且数量还在快速增加中。其带来的价值正在悄然中迅猛到来，已经成为前端开发人员必须掌握并提高自身价值的新技能。

如果说 2016 年没有掌握 Node.js 开发的前端人员会失去竞争力的话，那么 2017 年，没有熟练掌握微信小程序开发的前端人员，在前端开发人员队伍中，也一样会缺少竞争力。

本书特色与内容架构

循序渐进、由浅入深

作为一本开发者使用的学习参考书，本书讲解知识点时，遵从循序渐进的原则，将所有需要掌握的知识点做了系统化的组织和编排，每讲述一个知识点，均有相应的内容和案例解说，让读者在每一章节中都体会到自己的成长。

本书参考了官方文档的知识结构，为了使读者尽快入门，按照由浅入深的原则对章节进行了重新编排，帮助读者快速上手。

案例生动翔实，图示丰富

本书大部分案例都只针对相应的知识点，在完整介绍知识点的前提下尽可能精简内容和代码，读者阅读实例会感觉到非常轻松，学习知识点的时候，没有多余知识点分散精力，集中掌握小程序开发的目标，降低学习成本和理解的难度。

另外，在讲解知识点时，对不容易理解的地方书中使用了丰富的图表来展示，必要时用编号、标记等清楚地标记了操作的顺序和重点，让读者把更多的精力放在开发和实践中。

语言朴实，风趣幽默

虽然是一本讲解编程方面的图书，但本书并没有采用教科书式的刻板语言，

而是尽可能用通俗的语言，风趣地解读其中的内容，力保读者在轻松、愉悦的环境中完成学习。

适用读者和致谢

本书写作的目的是为了让所有对小程序感兴趣的人可以快速上手。

- 技术人员可以通过本书的技术开发章节快速了解小程序的开发以及调试方法；
- 创业者或产品经理可以从本书中迅速了解小程序的适用性以及优势；
- 运营人员可以迅速入门掌握运营规范以及应该避免的问题。

读者可以根据需要选择不同的章节进行阅读参考。

由于小程序更新频繁，本书中所介绍的开发接口以及工具版本可能并不是当前的最新版，在一些细节上与最新版本的小程序可能会有些不同，读者在具体动手开发时需要以官方的当前版本为准。

本书主要由沙拉依丁·苏里坦与黄曦创作，写作过程中得到了轻课 CEO 肖逸群的大力支持和鼓励，还有唐敬之、郑祝萍、顾立人、王启元、米昱杰、何川、刑菁、王超群和刘剑等人参与了编辑整理工作，以及轻课提供的平台与资源，在此表示由衷的感谢。并感谢家人、朋友们以及同事们一直给予的帮助和鼓励。

写作过程中难免有所纰漏，欢迎读者批评指正，并提出宝贵建议。

黄曦　沙拉依丁·苏里坦

2017 年 3 月

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **提交勘误**：您对书中内容的修改意见可在【提交勘误】处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **与作者交流**：在页面下方【读者评论】处留下您的疑问或观点，与作者和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/31331

目　录

第 1 章

小程序特点与开发逻辑

“我们希望存在一种新的公众号形态，这种形态下面用户关注了一个公众号，就像安装了一个 APP 一样。他要找这个公众号的时候就像找一个 APP，平时这个号不会向用户发东西，所以 APP 就会很安静地存在那里，等用户需要的时候找到它就好了。”

——张小龙

1.1 互联网正在变得越来越“轻”

2016—2017 年依旧是国内互联网环境复杂多变的年份：无论是“内容变现”崛起，还是 Papi 酱获得巨额融资，或是果壳憋大招的产品分答上线（如图 1.1 左），都引起了大众的强烈关注。

2016 下半年分享经济持续火爆，自行车共享新赛道（如图 1.1 右）引得各路资本巨头竞相押注。与此同时，我们注意到其中的每条消息绝无例外，都会在微信朋友圈掀起一阵刷屏之势，这似乎在强烈地印证着以微信为载体的信息传播方式已经成为主流的信息传播途径之一。现在微信本身已不再是一个单纯的网络通信工具，它已经成为了一个基于 8.06 亿月活跃用户（官方数据）群体的完整生态圈——从微商的对个人销售，到寻找企业资源的对企业的合作，各行各业都已在微信平台里尝试和摸索新的业务拓展方法。

微信的订阅号以及服务号体系早已被企业用户们所熟悉。通过合理合规的运营，企业将极大地降低获客成本，并为用户提供更加方便的使用途径。有非常多的优秀公司，尤其是 To C 的企业都在公众号开发上不遗余力，在微信的体系内实现自我宣传、一键下单、呼叫服务等，这都让消费者获得了相当流畅的使用体验。和下载一个几十兆甚至上百兆的 APP 相比，关注公众号就可以使用所有功能显得异常地省时省力，如图 1.2 所示为微信端浦发银行信用卡宣传广告和滴滴出行的下单界面。

图 1.1

图 1.2

另一方面，对初创型企业来说，相比开发一个 APP，公众号的对接流程相对简单，开发测试所需的时间周期短，前期投入的人力和物力成本也会相对较低。因此很多商业模式都可以通过微信公众平台以极低的成本迅速地被市场验证。

如图 1.3 所示为微信订阅号及服务号的登录后台。

图 1.3

作为基于微信生态圈的创业者可能对此更有深刻体会，不得不说，很多创业者在创业前期总是会掉入产品细节的陷阱。笔者认为，现在的爆款产品并不是投入十几个或几十个顶尖级工程师坐在一起研发十几个月就能得到的成果，而是在一个简单的产品模型或者设想投入市场验证需求后进行快速功能迭代的合集。从案例上来分析，无论是 Airbnb 还是 Uber，前期系统所提供的功能和设想都非常简单，成为“独角兽”的原因并不是一开始就给用户推出了重量级和体验都非常优秀的产品，而是抓住了用户的需求，并且通过后期功能的迭代和用户的增长不断积累到今天的量级。

创业的过程往往是一场和时间进行赛跑的游戏：尽快拿出早期的产品原型，投入市场进行验证，争取到投资，拿到第一桶红利，再进行迭代升级、融资、扩大市场份额，如此循环往复方能势如破竹般找到正确的发展方向，如图 1.4 所示为 Uber 早期版本和 2015 年的版本对比（美国版本，来源 http://thehustle.co/proof-that-your-favorite-startup-started-out-awful）。

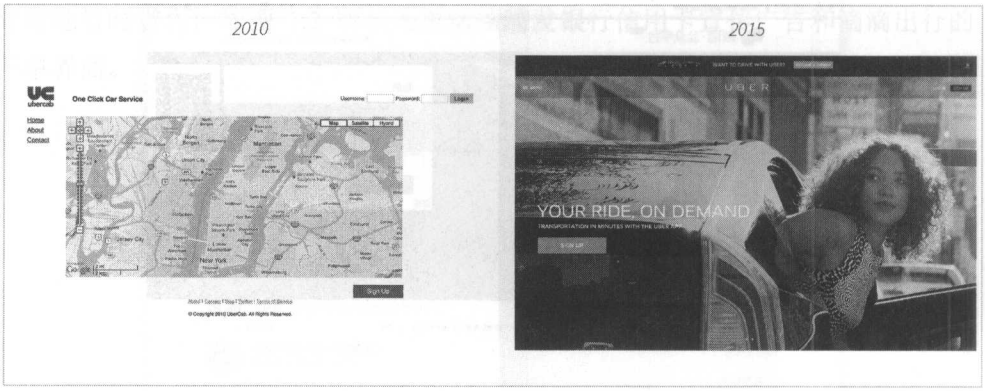


图 1.4

在此过程中，由于微信传播和获取的便捷特性，降低了整个产品需求验证的难度以及成本，所以应当考虑是否可以先暂时不开发 APP，而使用微信公众平台代替。当然，APP 的重要性无可厚非，不过在稳固市场份额之后，再投入成本研发 APP、建立品牌也并不算迟，并且这种“快准狠”的方式是对传统基于 APP 创业的同类型竞品基于时间维度的一次“降维打击”。

不过，和 APP 应用相比，基于公众号的应用并非没有硬伤。

首先，公众号基于 IM 对话完成功能，主要通过文字、语音等手段实现互动，并且支持跳转到第三方页面，但从用户角度来说，这个应用交互方法不够丰富。

其次，载入跳转第三方页面还只是基于 H5 形式展开业务，信息的承载方法（例如音频、视频等）较为单一，并没有脱离常规的 Web 形式，比起使用 APP 的体验仍有差距。

再次，公众号 API（Application Programming Interface，应用程序编程接口）接口对手机硬件调用能力较弱，虽然有基本的地理和拍照等组件可供使用，但是与独立 APP 对手机硬件模块利用度相比差距依旧较大。

从早些年开始，一直有传闻微信要推出独立的“应用号”体系。从名字上来猜测，应用号应该会像公众号一样让用户免安装，即关注后就可以使用，但是又同时提供类似独立 APP 应用的体验。2016 年 9 月 21 日，微信毫无征兆地向全行业 200 余家开发服务号的企业发出了“小程序”内测邀请。很幸运，笔者所在公司“轻课”也成为第一批内测资格获得者之一，使得我们有条件学习和掌握相关的规则。

可以预期的是，小程序的出现将会让互联网创业者们“脑洞大开”，新的玩法也将不断地被解锁。通过小程序的开发和使用，早期产品可以节约大量的人力和时间成本，基于 JavaScript 的实现也让基于移动端创业的门槛更低。前期产品完全可以使用小程序 + Node 后端来实现 JavaScript 全栈化，甚至可能出现做产品、开发、运营的独立创业者。由此看来，创业的方法、使用的平台、实现的技术在前期产品验证阶段会越来越成熟以及轻量化。从用户角度来看，应用获取和了解方式以及使用习惯也会变得越来越轻量化。

1.2　什么是小程序

那么，什么是小程序呢？它的发展又会对当前的 APP 应用商店有什么冲击呢？下面我们简要说明一下。

1.2.1　小程序的由来

2016 年 1 月 11 日，2016 微信公开课 PRO 版在广州举行，被称为“微信之父”的张小龙在首次公开演讲中说道：“但是我们的本意并不是要做成一个只是传播内容的平台，我们一直说我们是要做一个提供服务的平台，所以后面我们甚至专门拆分出一个服务号出来，但是服务号还是没有达到我们的要求，说服务号可以在里面以提供服务为主，所有的服务号还是基于一个诉求，这不是我们想看到的。现在我们将开发一个新的形态，叫作应用号。”

由此，应用号受到业内普遍关注，根据张小龙的描述，应用号的形态大致为：“我们希望存在一种新的公众号的形态，这种形态下面用户关注了一个公众号，就像安装了一个 APP 一样，他要找这个公众号的时候就像找一个 APP

一样，进去使用这个公众号，平时这个号不会向用户发东西，所以 APP 就会很安静地存在那里，等用户需要的时候找到它就好了，这样的话我们可以尝试做到让更多的 APP 有一种更轻量的形态，但是又更好使用，这是我们在探讨的一种新的公众号形态，叫应用号，这里只是提前剧透一点点东西。”

现在来看，小程序即 2016 年年初张小龙提到的“一种新的公众号形态”，也即应用号。估计是根据其定位的“一种更轻量的形态”，将其名称最后确定为小程序，而不是应用号。

1.2.2 小程序的发展与展望

小程序最主要的特点，是不需要下载和安装，信息触手可及，用完即走，无须卸载。

张小龙说：“小程序非常接近于 PC 时代的网站服务，网站服务不同于公众号，它更直接。大家想象一下，把小程序看作 PC 时代的网站可能更好理解。”

其实小程序与其说是 PC 时代的网站服务，不如直接说就是类似于网站的手机端服务，我们可以按照张小龙的建议想象一下，假如我们需要一个服务，比如买个衣服，我们打开网站，注册账户，填写收货地址，提交订单，选择微信或者银行卡支付，确认信息，完成支付，然后就可以把网站关闭了。

在此过程中，没有下载任何东西，购买完，我们就关闭了网站，也没有做任何停留。

小程序想做的，就是类似的场景，但比这还要简化很多，想象一下，如果是在小程序中，完成这样一个购买流程需要做什么？

下面以小程序京东购物为例来说明。

（1）在小程序入口处选择“京东购物”，进入京东购物小程序并允许授权登录（无须注册），如图 1.5 所示。

图 1.5

（2）搜索产品，以 vivo 手机为例，单击“立即购买”按钮。

（3）填写收货地址（如果在某个公众号服务中填写过，甚至在小程序中都不用再次填写），确认信息后提交订单，如图 1.6 所示。

图 1.6

（4）选择“微信支付”，支付完成后关闭小程序即可，如图 1.7 所示。

可以看到，小程序提供了更简化的过程，优于网站的体验，并且与网站一样即用即走，无须安装卸载。

图 1.7

但是，与网站不同的是，看到喜欢的网站，可以把网站放到收藏夹里，也可以记住其域名，下次直接输入域名来访问，但是小程序不提供这样的入口，我们只能通过扫二维码或者搜索来找到它。从这一点来说，小程序更有利于线下服务的线上拓展，比如来到一家餐馆点餐，我们就可以直接通过扫二维码的方式，访问其小程序，完成对喜欢的菜品的点餐，并直接支付下单，然后等待美味的菜肴即可。

这里张小龙自己也提到一个小程序的场景：

“现在汽车票其实没有电子化，所有人去坐汽车必须到汽车站现场去买一张票，这是一个很痛苦的过程，你要去排队买票，然后再去坐车，他们希望用小程序来解决这个问题，只需要在每一个汽车站立一个二维码，所有到汽车站的人扫一下二维码就启动购票的小程序，然后直接通过小程序来买好票，这样售票窗口就不用存在了，我认为这是一个非常贴合小程序的想法。”

通过这样两个场景，大家便能很清楚地了解小程序最适合的场景了，也因此很多人提出微信小程序会让 APP 消亡的观点，这其实是不实际的，小程序的目的也并非如此。微信小程序是希望通过自己轻量、即用即走的特点，来拓展 APP 的功能，希望 APP 的开发者们拿出一些很核心的功能做得更加精细化，并且能够与线下的场景实现更紧密的连接，体现了微信“连接人与服务”的愿景。

未来微信小程序的发力点就在于解决这类场景，仔细想来，现实中其实这类

需求空缺是非常大的，由于没有一个统一、规范的入口，虽然可以通过 H5 等技术实现，但是因为流程始终无法简化，对用户而言始终是比较复杂的。通过微信强大的平台入口，按照其平台规范，以及简化的流程和操作，终将解决这个问题。

1.3　你的产品适合做小程序吗

微信官方对于小程序的定义是：微信小程序是一种不需要下载安装即可使用的应用，它实现了应用“触手可及”的梦想，用户扫一扫或搜一下即可打开应用，并且即用即走。那我们就从这个描述说开去，来探讨究竟什么类型的产品适合开发小程序。

大家都知道手机对于数据的存储和处理能力相对有限，市面上利用手机本身的硬件资源去做大量计算处理的 APP 非常少，除了图像处理或者需求实时解码的情况，数据运算和逻辑处理大多数情况下都交于服务端进行处理并返回给 APP 端。在企业级 APP 应用中，APP 端更多是充当着 VIEW（展示）层角色，小程序也是同样的道理。小程序的设计并不是为了去完成复杂的逻辑运算，获得更高的负载，这些问题应该是后端工程师去考虑的问题。从产品角度上来说，小程序是一种方便用户分享传播、获取以及使用的展现方式。

小程序本身是基于脚本完成的，相较原生 APP，小程序的性能指标并不会很快，同时基于封装组件的模式也并不适合去开发业务逻辑非常复杂的产品以及界面。但是，小程序的存在弥补了轻量级微信公众号模式以及重量级 APP 产品两者跨度太大的问题，让功能模块化应用有了合适的载体。

因此，对于企业来说，小程序可以用来封装现有模块作为独立的产品。例如，在很多教育类的互联网产品中，帮助用户养成良好的学习习惯是非常基础的一环，同时它也是提升用户活跃度和黏性的重要手段，这就产生了打卡签到类的功能模块，甚至市面上已经有专门针对这一模块的独立 APP 了。其实在实际使用的过程中，根据笔者观察，很多用户都会认为如果使用独立的 APP 进行打卡签到实在太重量级，基于业务的打卡签到（类似 Foursquare 早期形态）也会让人感到产品功能不明显，重点不突出，但是如果把打卡签到独立到小程序应用中就比较合适了。首先，打卡签到属于轻量级但是高频次而且功能相对单一的应用，基于微信小程序的设计会让打卡提醒、打卡方式都有所缩减，整个产品体验会更好，功能点更

突出。第二，打卡应用的产品体系是符合传播逻辑的，用户在打卡完成任务之后更希望去分享，所以可以通过微信直接分享给朋友或者发送到微信群。

2012 年 8 月，硅谷创业家 Eric Rise 在其著作《精益创业》（*Lean Startup*）一书中提出“精益创业”这一概念。其核心思想是，先在市场中投入一个极简的原型产品，然后通过不断地学习以及有价值的用户反馈，对产品进行快速迭代优化，以期适应市场。这一理论一推出便赢得了许多创业者的赞同，在中国更是广为传播，最小化可行产品（Minimum Viable Product，MVP）就是其中的核心工具和观念。

简而言之，这一理念就是把复杂的需求精简到最核心的点，使用有限的资源去验证核心的产品功能。比如，如果产品的定位是一部汽车，那我们首先需要去验证“移动”这个需求是不是被市场所接受的需求。从设计一个最简单的可以帮助用户去移动的工具开始，比如一个滑板。滑板证实了“移动”这个需求是普遍的，表示本产品是可行的，我们再迭代加上动力装置（引擎），防护装置（车体车门），以及其他让用户感觉到方便的功能（例如加热坐垫等），最后再完成一个独立的产品。我们可以把制作滑板的成本理解成试错成本，在这期间就算需求不被接受，对于创业者来说损失也不会很大。比较而言，如果我们把试错成本放在最终产品上，一旦需求验证错误损失将会很大。

笔者在前面曾经描述了 Airbnb 和 Uber 从基础原型产品做到爆款产品的例子。事实上这种例子可以被描述成“滚雪球”式的创业方法。Justin Vincent 在 *Don't Start Big, Start a Little Snowball* 文章中也说明项目的起始点不用做得很复杂，从最小化去验证产品可行性。Uber（最早叫 UberCab）最开始积累只是在私人朋友之间尝试使用，最初的定位并不是使用复杂框架和优美 UI 去解决问题，最早的方法只是尝试去看看大家的喜爱度，结果一个月之后，用户量翻了十倍证明这是一个可行的方案。

Airbnb 的创始人 Brain Cheskey 和 Joe Gebbia 的最初设想就是“Air Bed and Breakfast”，提供便宜的充气床以及一顿方便快捷的早餐。经过市场验证可行之后，他们延展了商业模式，开始提供出租房间以及公寓。最初的产品想法甚至单纯到有点无聊，但是经过后期的验证却获得了巨大的成功。这个最简单想法的提出，在周围人看来甚至有点可笑，但是他们却不是需求的制定者，任何想法无论多伟

大，或者多低微，都必须经历市场的验证之后才能评价。如果读者觉得这两个示例还是太过于偶然，只能展示出最近的创业独角兽级的公司的小概率事件，根据 Justin 的文章，我们可以去深挖 Google、Slack、Dropbox 的历史，无独有偶，它们开始时都是通过简单的产品去尝试一个最基本的功能，并且把“雪球”慢慢滚到了今天的这个体量。决定从 0 到 1 的是市场需求，决定从 1 到 N 的是增长和运营，验证最小化可行产品是从 0 到 1 阶段最重要的调研。

小程序的出现使得验证最小化可行产品开始变得高效。此前，在互联网上广为传播的《别再开发 APP 了》一文的作者也提出了相同的观点，“APP 生态已经趋于饱和，低频产品已经没有开发 APP 的必要，转而开发微信公众号（未来是小程序）将是最佳选择。”在产品验证期间开发 APP 的成本太高，许多创业公司也都死在 APP 研发上：从开发的成本，更新周期的等待，到质量的把控，其中每个环节都有可能把创业公司扼杀掉。当然，也有很多人对这一观点并不认同，曾有人跟帖留言说：“如果我总是依赖于微信，我们的用户数永远不可能比微信多”。笔者认为，这个说法是正确的，但是你可能需要先去调查一下微信总用户当前有多少，并且看一下比这个用户数多的 APP 有哪些。如果某天你真的做到了这个体量，导流到 APP 端或者 Web 端对你而言肯定易如反掌。

不得不承认的是，小程序非常适合前端工程师走全栈路线。前期可能仅需一个熟悉 Node 的 JavaScript 工程师就可以挑起整个项目。目前移动化趋势也越来越偏全栈化，从前几年基于 PhoneGap 的应用火热，到现在安卓与 iOS 应用都可以使用 React Native 实现，一种语言做多个平台已经超出了 Java 等其他语言“跨平台”的意义。此外，小程序的理念和 React 基于虚拟 DOM 的设计思想如出一辙，这也是 Web 前端技术的一个进化的方向。

小程序项目团队组织架构和具体的项目需求挂钩，但这并不是说微信公众号或者小程序的技术含量就比其他实现方法要低。所有的展现层只是冰山一角，冰山在水下面的体积永远让人无法估计，笔者也曾经经历过一个公众号有几十万并发的场景。在项目执行前我们就需要对用户增长量有一个预判，我们需要去判断这个应用是一个指数级跳跃还是一个线性增长的项目。

用户数以指数级跳跃增长的应用经常被称作“爆款应用”，如果你的应用十分幸运地成了爆款，那恭喜你，希望你之前系统搭建时就已经考虑到了高负载和高扩展性。虽然大多数项目都是线性增长，尤其对创业项目而言，但无论你的产品

“爆”与“不爆”，后端的架构都可以不断优化和替代。基于小程序的初创团队可以选择使用前端、后端独立开发的模式搭建技术团队，如果条件合适，也可以使用 Node.js 进行后端处理，前端配合小程序开发，通过 Restful 等协议进行通信，这样整个技术团队可能就可以精简到一个人。因此，对于功能简单的 Demo，成本是不是一下子降了不少？

微信在 6.3.7 版本中下线了下拉拍摄视频功能曾经引起不小的争议，各种“原因”在网上传得沸沸扬扬。很多人说这是因为这个低频次的功能打扰到用户的正常操作所以被取消了，其实这也和小程序的产品设计与布局理念一样，应该遵循“不打扰用户”的核心设计原则。在微信官方的小程序设计指南中特别指出小程序应该减少使用无关的元素并且降低干扰。相对地，小程序应该使用基于微信生态体系的布局标准，建立起让用户可以理解、接受和突出重点的布局方法。对于让用户感觉到迷惑的界面布局应该尽量避免，设计者应该遵循各种控件的使用规则以及场景。页面也应该尽量突出本页面的主要重点。官方文档演示了一些典型的错误布局方法，这些错误的设计方法也有可能是之后因为布局风格不符合规范而被拒绝通过发布审核的原因。另外，产品设计时应该遵循官方的设计规范，例如，在 APP 应用中开发者可以定义程序的启动画面，小程序启动画面由系统完成，开发者并不可以自己更改，启动的页面上包含程序的 Logo 以及加载进度。

1.4 小程序特色：即用即走

小程序从最初的产品设计时就定位为即用即走，它是一个轻量级的产品的设计理念。即用即走并不是说小程序的入口是不存在的。正相反，小程序的入口是在比各种线下途径最容易接触到的地方。公交车站的班次表、餐厅的桌面上，甚至医院的挂号窗口等都可以成为小程序的入口。线上传播可以通过一对一的聊天框和群组群聊分享任意一个小程序页面。

微信是一个专注于产品的团队，从产品的定位，到消息提醒打扰的次数设定，一切都是从用户的角度来考虑的。对于后期运营者来说，一些功能是反常规并且很难被接受的，但对于用户来说确实是增强了产品的体验，“即用即走”就是其一。

罗振宇在逻辑思维 2016 年跨年演讲中提出了时间占领的概念。他认为有限的

时间比上无限的信息结果是零。在将来，用户黏性可以从用户在本产品上花费的时间来计算。从运营层面上来看，用户如果愿意在产品上花更多的时间，用户的付费转化的概率也会相应地提高。微信希望小程序出现在降低用户黏性、提高用户体验的场景，所以他并不希望去占领用户的时间。所以与公众号后台比较，小程序并不是使用关注的粉丝数，而是用使用次数作为单位进行统计。

电商和游戏是典型的时间占领的产品。微信团队已经说明小程序将不支持游戏，而且小程序不提供线上入口，电商也较难发展运营。在线下，各种小程序通过带参数的二维码可以快速传播，可以预测的是，第二轮的二维码爆发是基于微信小程序的。越来越高的线上获客成本，将导致开拓线下获客渠道可能是新一年互联网流量的来源。相比 APP 端，基于微信的扫一扫可能是最方便的扫二维码软件，如果直接结合新推出的小程序，可能让线上线下玩法出现一个新的高峰。

1.5　小程序与订阅号、服务号的异同

微信推出了小程序之后，是不是就不需要去使用订阅号以及服务号了呢？答案是否定的，我们需要从微信公众平台的账号类型去了解公众号体系的特点以及所提供功能。目前微信公众平台提供四种账号类型：小程序、订阅号、服务号以及企业号，小程序应用场景在之前章节已有详细介绍。

- 订阅号：主要用于需要频繁给用户传递内容类信息的场景，例如新闻订阅、杂志更新、每日热点等。订阅号推送消息较常用的展现形式是图文消息，用户通过点击所推送的图文消息链接可以外链跳转到具体帖子或者其他网页地址来实现内容阅读。订阅号每天只允许对所有用户进行一次群体推送（简称群推）。订阅号对个人申请者开放，虽然不能使用开发类接口，但如帖子编辑、内容编辑类功能，以及用户交互、数据统计等功能均可使用，需要注意的是订阅号可以升级成服务号，但是服务号是不可以转成订阅号的，如图 1.8 所示为一个朋友（微视角）的订阅号内容。

图 1.8

- 服务号：应用场景偏向于信息提醒类型的场景，例如用户消费提醒，购买支付成功提醒。服务号也和订阅号一样可以群发消息，但是每个月只有 4 次机会。相比订阅号，服务号开放了更多开发接口，允许公众号对关注的用户发送模板消息进行提醒，并且开放了二维码生成接口（临时和永久二维码）等，如图 1.9 所示为“我的印象笔记”的服务号。
- 企业号：经常被用于企业内部通信，并且只有验证之后才可以关注，如图 1.10 所示为一家公司的企业号。

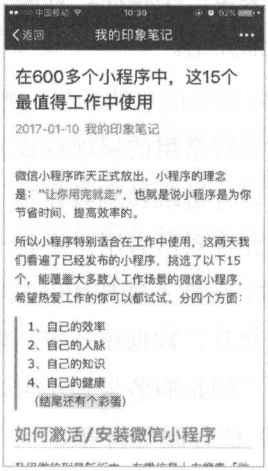


图 1.9

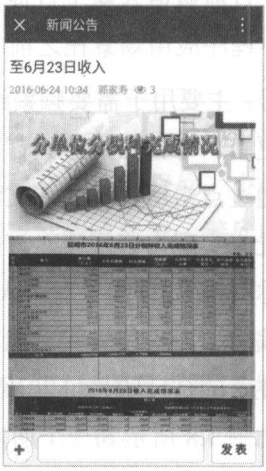


图 1.10

微信公众平台并不提供用户真实数据（微信号、电话联系方式等），这也就意

味着如果有用户关注了你的公众号，你并不能拿到用户的详细信息以及联系方式。系统上的用户识别方式通过 OpenID 的形式实现。OpenID 是微信分配给每一个用户在某个公众号中的身份唯一识别码，它通过某种不可逆的哈希加密算法计算来获得（可能是公众号 ID+用户微信号进行了哈希运算），这个号码不会根据用户取消关注或者再次关注而改变。运营者应该妥善保管用户 OpenID 信息，并且防止信息的泄露，任何人都可以在知道了公众号的对接信息（Access Token）以及用户的 OpenID 之后给用户发出类似于发消息提醒、拉黑、设置独立自定义菜单等操作。在多个微信公众平台账号体系之间，可以使用 UnionID 来关联相同用户在不同账号之间的身份关系。UnionID 通过微信开放平台（区别于微信公众平台）绑定不同的公众号、应用等来鉴别同一个用户身份的设定。在多个公众号绑入同一个开放平台的账号后，微信会对用户增加提供额外的 UnionID 字段来标识用户在不同账号中的关联性。

在具体业务使用中，如果某个用户在公众号 A 中购买了某种功能，但需要在公众号 B 中使用，因为两个公众号用户的 OpenID 不一样，是无法使用的，这时因为管理员从开放平台注册并且关联了两个公众号，所以 UnionID 是相同的并且产品应该使用 UnionID 作为用户的唯一标识而不是 OpenID。又例如，如果产品只是涉及一个公众号做用户的标识，这时使用 OpenID 作为用户的识别即可，并不需要使用 UnionID。

笔者推荐保存公众号所提供的用户信息到本地数据库中，包括 OpenID、头像地址、地区、性别、语言、关注状态等，有以下几个原因。第一，除了用户改变昵称以及头像的情况之外，用户的信息更新并不是很频繁，本地检索用户信息获取速度要比通过 API 向微信检索更快，有利于用户体验。第二，用户的关注状态可以通过公众号发的 event 消息类型进行更改，用户关注和用户取消关注行为都会触发微信系统对后台对接的接口发送此事件类型的通知，在后台我们可以统计到用户取消关注的具体状态以便做留存等行为分析。第三，用户获取接口每天是有限的，虽然当前调用可能并没有达到每日上限，我们还是需要在设计时留意系统的可扩展性，能用缓存的信息使用缓存，尽量减少 API 的调用次数。系统可以每隔一定周期使用微信批量获取用户信息 API 来更新所有本地保存的用户信息，当然按照用户行为来触发更新信息也是可行的方法，只是可能本身用户信息并没有更改，并不需要此时更新，造成了数据获取、资源计算的浪费。

1.6 消息推送与传播分享

订阅号和服务号消息推送是公众号和用户交互，带来日活，获得留存的基本方式。公众号和用户短期之内的交互都可以使用客服消息来完成，但通过 API 接口发送客服消息时，需要用户与该公众号 48 小时内有互动。如用户 A 给公众号发了一条文字消息，这时候公众号可以在 48 小时之内通过接口回复该消息，如果用户之前没有主动发送消息，或者互动超过了 48 小时，调用接口将会显示错误信息。

模板消息是服务号以及小程序的消息提醒方式。发送模板消息类似手机发送短信，并没有交互时间的限制，用户只需要关注并且打开消息功能就可以接收服务号发送来的模板消息。模板消息内容格式是固定的，公众号需要按照需求选取官方提供的模板进行发送，发送时候公众号只允许更改信息内容字段。公众平台为了防止模板消息滥用，给用户造成骚扰，严格限制了模板消息发送的每日次数（10 万，100 万，1000 万关注粉丝量所获得的每日可发送的模板消息和客服消息数不同）以及单个用户同时可以收到的信息总条数。

在小程序中，开发者可以给用户推送模板消息，但是仅限于支付和填写表单应用场景。在用户接受服务之后，7 天内可以给用户发送一条模板消息作为提醒。可以看出，相比公众号在模板消息上，微信严格限制了信息推送能力，这就导致许多时间占有类的应用场景并不适合在小程序上进行实现。

客服消息在小程序中的体现为即时性呼叫消息互动。微信规定客服消息只允许在互动后 48 小时内回复使用，并且客服消息将会被集中收纳到小程序客服消息窗。开发者可以选择将客服消息转交给多客服系统处理或者自行处理消息。

1.7 普通用户怎么玩转小程序

小程序优势这么明显，除了开发者外，普通用户又该怎么玩呢？下面简单介绍一下。

1.7.1 普通用户启动小程序方法

小程序的主要入口是二维码，或者其他分享者发来的小程序分享页，如果用户

没有通过二维码或者分享页启动过小程序，则在微信中不会有任何入口，如果启动过小程序，则会在微信“发现”页最下面一栏出现“小程序”入口（如图 1.11 左所示），点击进入，则可以看到自己启动过的小程序，以及一个搜索栏，用户也可以从这里搜索自己需要的小程序来进行启动（如图 1.11 右所示）。

图 1.11

1.7.2　普通用户在小程序里面能做什么

普通用户在小程序里，按照现有的 100 多个小程序，可以完成很多常见的功能，如我们所熟悉的美团外卖、滴滴出行 DiDi、摩拜单车等很多平常高频使用的大部分应用，都已经有了小程序。基本涵盖购物、旅行、票务、投资理财、工具、教育培训等各个方面，方便用户使用。

第 2 章

微信小程序开发申请入门与环境搭建

做过公众号开发的用户对微信开发流程一定不会陌生，小程序与其类似，本章我们讲述如何申请小程序开发，以及开发环境的搭建。

2.1 小程序申请方法以及流程

小程序作为微信公众平台的一员，申请方法与之前的订阅号和服务号类似。首先，申请者需要使用未在公众平台注册过的电子邮箱地址进行申请。在公众平台点击选择使用小程序账号并且进入注册流程，如图 2.1 所示。

目前小程序的开放注册范围包括企业、政府、媒体以及组织。在认证注册邮箱之后，会进入机构的确认流程，用户需要选择组织类型并且填写相关机构信息。以企业为例，申请者需要以企业为主体上传营业执照扫描件并通过打款验证对公账户。需要注意的是，微信公众账号（包括小程序，订阅号，以及服务号）属于注册认证后的主体，个人管理员以及运营者虽然有账号的使用权，但并不代表有账号的所有权。个人开发者借用其他企业机构信息代为注册时，虽然后期运营或者开发是个人完成，但账号可以通

过机构的申诉返还给所属主体。之前曾有过因为公众号主体所有权不同引起的纠纷案例。微信公众平台的体系功能区分明确，以个人为主体的订阅号不可以使用任何类型的开发接口，企业或机构为主体的订阅号可以使用部分开发接口（客服消息类型为主），但是无法使用模板消息（类似短信消息提醒）的接口，具体区别可以参考本书后续章节。

图 2.1

在账号开通之后，用户可以通过绑定的手机号扫码登录系统，并且填写相关小程序信息。目前公众平台通用的体系是通过管理员的运营者模式来管理账号。在输入注册邮箱以及账号密码之后，通过微信扫二维码的方式进行登录。在开发的过程中，如果其他人需要登录账号，可以告知账号以及密码，但登录时需要绑定的管理员或运营者的微信号验证授权，如图 2.2 所示。

小程序的子账号类型分为管理员、开发者，以及体验者。管理员账号是在本公众账号申请时绑定的微信号，开发者类型账号是使用本账号权限进行小程序开发的授权开发工程师账号，体验者账号是针对项目测试运营时为内测用户提供的

账号权限。目前管理员账号只允许存在一个（后期可以修改绑定），开发者账号和体验者账号的数量限制分别是 10 个和 20 个，如图 2.3 所示。

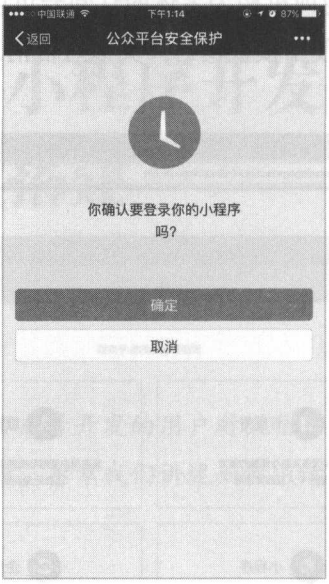


图 2.2

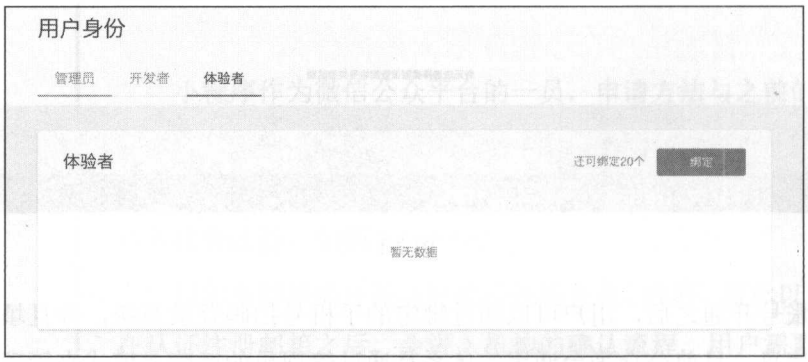


图 2.3

2.2 小程序开发环境搭建

小程序开发环境搭建相对简单，开发者前往首页的项目介绍页面，点击开发工具链接进入获取微信 Web 开发者工具即可。目前小程序开发工具支持 Windows

32/64 以及 Mac 平台，用户可以根据需要选择。需要注意的是，微信 Web 开发者工具目前更新速度快，每个版本之间添加的功能较多，开发者最好经常查看更新日志，了解最新功能并且更新工具包版本。

和 Eclipse 等开发工具不同，微信 Web 开发者工具需要扫码登录后使用，绑定微信账号需要授权到开发者权限之后开发工具才可以使用。在创建新项目时，开发者首先需要用从后台获取到的 AppID 来关联项目。在没有使用关联 AppID（开发体验者模式），开发工具的提交等功能将不能使用，如图 2.4 所示。

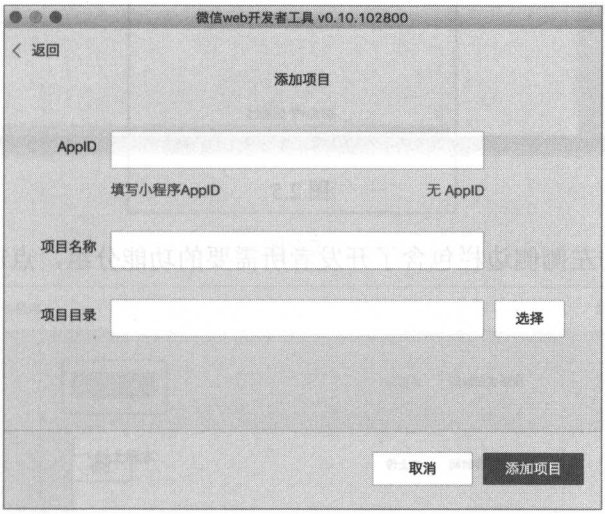


图 2.4

使用过微信 Web 开发者工具开发 JSSDK 的朋友应该对这个工具布局并不陌生，这是基于 Chromium 浏览器内核研发的编码以及调试环境，前端开发者可以沿用之前所习惯的网页调试方法来开发小程序。微信 Web 开发者工具提供了完整的研发和发布流程。从编码到模拟调试，从真机运行到项目发布都可以在这个工具中完成，如图 2.5 所示。关于微信 Web 开发者工具的详细介绍请参考本书后续章节。

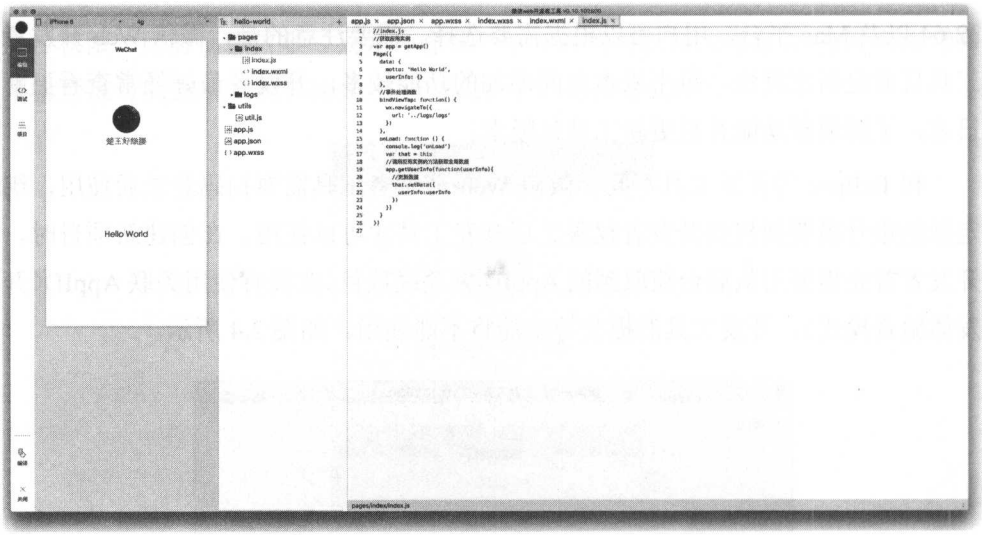


图 2.5

在上图中的左侧侧边栏包含了开发者所需要的功能分组，点击如图 2.6 所示预览按钮。

图 2.6

在系统提示手机扫码之后，即可在手机端运行，如图 2.7 所示。

小程序发布过程的第一步是基于微信 Web 开发者工具完成。小程序开发完成之后可以在项目页面中点击上传，当前项目将会滚入开发版本中，这时候可以对版本进行提交审核，在通过审核之后，在项目首页上点击“发布”按钮即可发布，如图 2.8 所示。


●●●○○ 中国移动
14:48
67%
WeChat
煮熟的米
Hello World

图 2.7

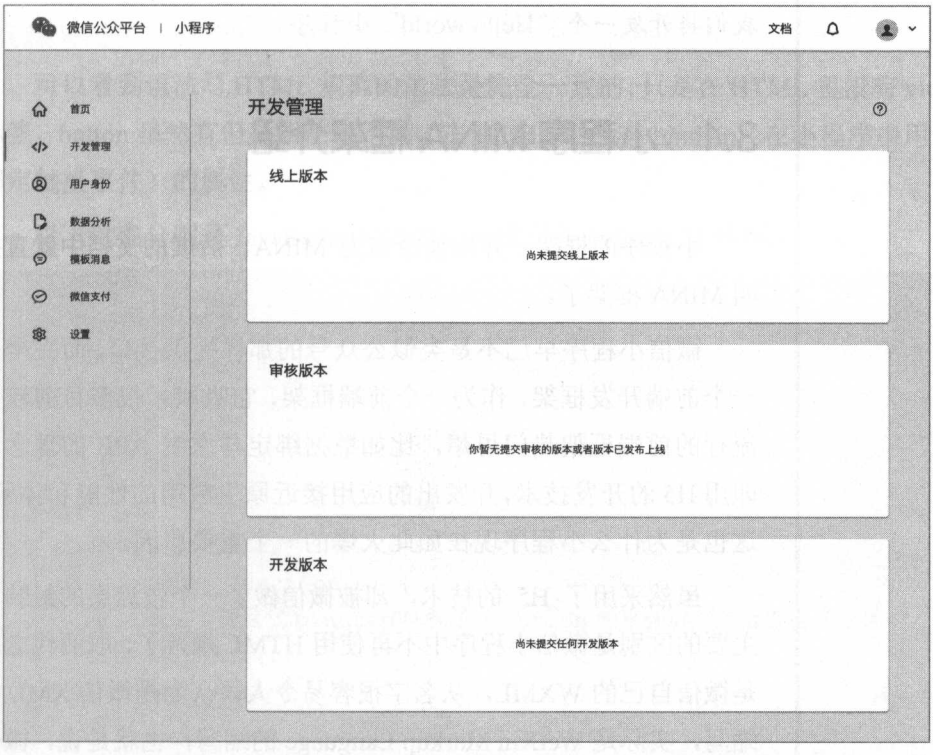

微信公众平台 | 小程序
文档
首页
开发管理
用户身份
数据分析
模板消息
微信支付
设置
开发管理
线上版本
尚未提交线上版本
审核版本
你暂无提交审核的版本或者版本已发布上线
开发版本
尚未提交任何开发版本

图 2.8

第 3 章

初识微信小程序：小程序的 Hello World

本章，将开启我们的小程序之旅。通过本章的学习，我们将熟悉用于小程序开发的Web微信开发者工具的使用，并按照惯例，我们将开发一个“Hello world”小程序。

3.1 小程序 MINA 框架介绍

小程序的框架一开始被命名为 MINA，后续的文档中就直接叫 MINA 框架了。

微信小程序早已不是类似公众号的那样提供接口，而完全是一个前端开发框架，作为一个前端框架，它吸取了很多目前较为流行的前端框架热门思想，比如数据绑定与原生 APP 的理念，即用 H5 的开发技术，开发出的应用接近原生应用的性能和体验，这也是为什么小程序现在如此火爆的一个重要原因。

虽然采用了 H5 的技术，却被微信做了一个较深层的封装，主要的区别是微信小程序中不再使用 HTML 规范了，取而代之的是微信自己的 WXML，从名字很容易令人误认为是微信 XML 的缩写，实际是 WeiXin Markup Language 的缩写，也就是说，微信把它当作是一种新规范，不过因为其语法和HTML规范完全一致，

并不会给开发者增加难度。而小程序的样式控制，则叫 WXSS，全称 Weixin Style Sheets，但目前而言，WXSS 除了名字与 CSS 不同外，其语法甚至用法都是和现有 CSS 标准完全一致的。需要注意的是，小程序完全不兼容 HTML 规范，也不存在浏览器 DOM 以及 BOM 的概念，因此在 JavaScript 中，也无法使用 Window 对象，熟悉 Node.js 开发的同学对此也应该不会陌生。因此自然也无法使用 jQuery、Zepto 之类操作浏览器 DOM 的现有框架。不过喜欢使用 jQuery 与 Zepto 框架的同学也不用担心，微信小程序使用时下最流行的 View 层数据绑定的概念，已经在很大程度上简化了开发，可以让开发者把精力聚焦于数据与逻辑上。

对于使用框架的数据绑定，官方文档提供了如下示例代码。

以下这部分就是上述提到的 WXML 文件的内容，我们称之为视图层，通常写在 index.wxml 文件里：

```
<!-- 视图层，即写在 index.wxml 里的内容 -->
<view> Hello {{name}}! </view>
<!-bindtap 用于绑定 changeName 事件到组件上 -->
<button bindtap="changeName"> Click me! </button>
```

可以看到语法与 HTML 规范的语法是完全一致的，只是在 HTML 里没有 view 标签，button 虽然有但是用法也不同，在 HTML 中没有 bindtap（在小程序中用于绑定触摸事件）的属性。

以下这部分则是 JavaSctipt 的内容，通常写在 index.js 文件里：

```
// 逻辑层
// 数据与事件绑定
var helloData = {
  name: 'WeChat'
}
// 注册一个页面
Page({
  data: helloData,
  changeName: function(e) {
    // 注意这里是通过 bindtap 实现了与 button 的事件绑定，后面会详细讲到
    this.setData({
      name: 'MINA'
    })
  }
})
```

使用过 React Native 和 Vue 框架的同学可能会发现，这两种写法基本上是类似的，而且与 Vue 框架的写法相似性更高，微信通过一些限制，进一步简化了代码。

上述例子主要做了两件事：

（1）在视图层（即 index.wxml 文件）里的 name 与逻辑层（即 index.js 文件）的 name 进行绑定，在视图层用{{name}}指定数据绑定，并通过在逻辑层 Page 方法中传入带有 data 字段的参数对象来向视图层传入绑定的数据内容。

（2）使用 bindtap 属性在视图层绑定 changeName 事件，并通过逻辑层中的 Page 方法中传入带有 changeName 方法的参数对象实现该事件的调用。

而在 changeName 事件中，又调用了默认的 this.setData 方法来改写视图中 name 绑定的数据内容为“MINA”，即实现了点击按钮后，把界面中绑定的 name 的默认内容“WeChat”修改为“MINA”。

最终这个示例显示的内容由‘Hello WeChat’变为‘Hello MINA!’。

从这里可以看出这个框架在内部的代号应该仍为 MINA。

在这个例子中，也许你也注意到了，由于没有采用任何样式布局，因此没有创建 WXSS 文件。

3.2 小程序基本结构

图 3.1 所示是用微信 Web 开发者工具在创建空项目时自动生成的 Quick Start 项目代码的目录，其中包含一个完整微信小程序的结构，接下来一一详述。

首先是最外层的 app.js、app.json、app.wxss 文件，app.js 是整个微信小程序的入口文件，用来控制小程序的整体逻辑。

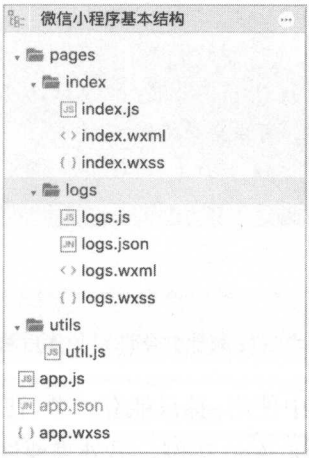


图 3.1

在 app.js 中，默认会调用 App()方法，可以传入 object 参数，其中有三个方法会被微信小程序自动调用，并在小程序中担任生命周期函数的作用，分别为：

- onLaunch，当小程序初始化完成时被调用，该方法只被调用一次。
- onShow，当小程序启动，或从后台进入前台显示时调用，因此可能会在每次进入前台显示时被多次调用。
- onHide，当小程序从前台进入后台时被调用，因此在每次切入后台时都会被多次调用。
- test，作为全局函数来调用，不会被自动调用。

实现完整的小程序生命周期的代码如下：

```
App({
  onLaunch: function() {
    // 当小程序初始化完成时被调用，该方法只会被调用一次
  },
  onShow: function() {
     // 当小程序启动，或从后台进入前台显时调用，可能会被多次调用
  },
  onHide: function() {
     // 当小程序从前台进入后台时被调用，可能会被多次调用
  },
onError: function(msg) {
//发生错误时调用，通过打印 msg 对象可以了解错误信息
   console.log(msg)
```

```
  },

  },
//相当于对象里的一个属性，可供全局使用
globalData: 'I am global data'
})
```

还有全局方法：

```
getApp()
用于获取小程序的实例，在之后的例子中会讲到具体用法。
```

App()方法必须在 app.js 中使用，且只能有一个，在 App()方法无须调用 getApp()方法，直接使用 this 即可访问 App 实例。另外需要注意在 onLaunch 方法中不能调用 getCurrentPage()方法，因为此时页面还没有生成，这一点我们会在后面的“手把手教你做 Demo”中体会到，也不能在 getApp()获取的实例中调用上述的生命周期函数，这些在官方文档中有特别说明。

而在 app.json 中，是用来全局配置小程序的，主要的配置项有 pages、window、tabBar、networkTimeout 和 debug，下面简要说明。

pages

pages 是一个数组，用来告诉小程序由哪些页面组成，每一项的名称对应页面的“路径+文件名”，第一项代表小程序的初始页面，只有在这个数组中指定的页面才会被小程序识别。

以图 3.1 所示目录结构为例，需要在 app.js 中写：

```
{
  "pages":[
    "pages/index/index"
    "pages/logs/logs"
  ]
}
```

window

window 是一个 Json 对象，用来设置小程序的状态栏、导航条、标题和窗口背景色，依然以图 3.1 所示目录结构为例，app.json 中的内容可以改为：

```
{
  "window":{
```

```
    "navigationBarBackgroundColor": "#ffffff",
    "navigationBarTextStyle": "black",
    "navigationBarTitleText": "WeChat",
    "backgroundColor": "#eeeeee",
    "backgroundTextStyle": "light",
    "enablePullDownRefresh": "false",
  }
}
```

为了便于理解，可以参照官方文档中如图 3.2 所示的界面。

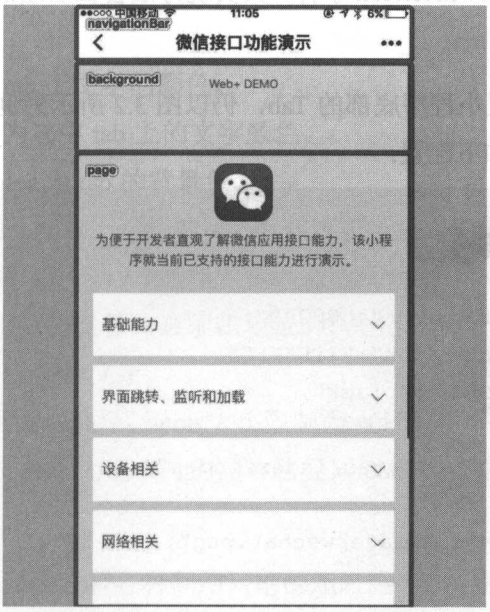


图 3.2

从图 3.2 所示中就可以看得比较清楚了，简要说明如下。

- navigationBar 主要指图 3.2 中的标题栏。
- navigationBarBackgroundColor 是标题栏的颜色，需要以十六进制色码如 #000000 格式指定。
- navigationBarTextStyle 是标题栏文字风格，这里只能从 black 和 white 中选择一个。
- navigationBarTitleText 是标题栏文字内容，如上述代码示例中的“WeChat”会显示在标题栏中。配置时注意背景色和文字颜色需要相互搭配，且不能

相同，如果把 navigationBarBackgroundColor 设置为#ffffff(白色)，再将 navigationBarTextStyle 设置为 white，那么标题就无法看到。

- backgroundColor 为图 3.2 中 background 区域的颜色，也需要以十六进制色码如#000000 格式指定。
- backgroundTextStyle 为图 3.2 中 background 的文字颜色，也只支持 dark/light，同样需要注意与背景色搭配且不能与背景色相同。
- enablePullDownRefresh 用来实现是否开启下拉刷新，设置为 true 会触发页面的 onPullDownRefresh 事件，其实现方法会在后续的章节做进一步介绍。

tabBar

tabBar 用来配置小程序底部的 Tab，仍以图 3.2 所示为例，可以在 App()方法的参数对象中增加如下配置：

```
{
    {"tabBar": {
   "color":"#000000",
    "selectedColor":"#000000",
    "backgroundColor":"#FFFFF",
    "borderStyle":"black",
   "list": [{
     "pagePath": "pages/index/index",
     "text": "首页",
    "iconPath": "image/wechat.png",
    "selectedIconPath": "image/wechat.png"
   }, {
     "pagePath": "pages/logs/logs",
     "text": "日志",
    "iconPath": "image/wechat.png",
    "selectedIconPath": "image/wechat.png"
   }]
  },
}
```

结合如图 3.3 所示的官方文档，就能清楚地看出解配置方法。

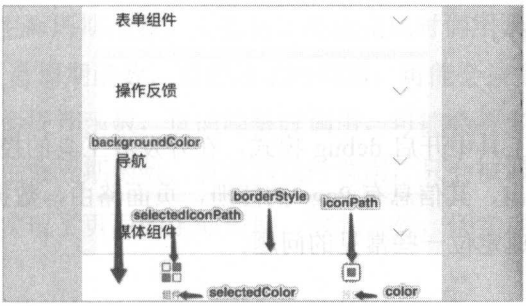


图 3.3

简要说明如下。

- 其中 color 为 tab 上文字的颜色。
- selectedColor 为选中 tab 上的文字颜色。
- backgroundColor 为 tab 的背景色。
- borderStyle 为 tabbar 上边框的颜色，仅支持 black/white。
- list 为 Tab 页内容，为一个数组。最少需要些两个，其中属性分别如下。
  - pagePath 为页面路径，必须在 pages 中先定义。
  - text 为 tab 上按钮文字。
  - iconPath 为图片路径，icon 大小限制为 40kb。
  - selectedIconPath 为选中时的图片路径，icon 大小限制为 40kb。

networkTimeout

networkTimeout 用于设置各种网络请求的超时时间，示例代码如下：

```
"networkTimeout": {
    "request": 10000,
    "downloadFile": 10000,
     "connectSocket": 10000,
     "uploadFile": 10000
},
```

代码说明如下。

- request：wx.request 的超时时间。
- downloadFile：wx.downloadFile 的超时时间。
- connectSocket：wx.connectSocket 的超时时间。
- uploadFile：wx.uploadFile 的超时时间。

其单位均为毫秒。

debug

可以在开发者工具中开启 debug 模式，在开发者工具的控制台面板，调试信息以 info 的形式给出，其信息有 Page 的注册，页面路由，数据更新和事件触发，可以帮助开发者快速定位一些常见的问题。

示例代码：

```
"debug": true
```

app.wxss 用来实现小程序的全局样式。

该文件作用是定义全局样式，作用于每一个页面，而每个页面的样式则只作用于该页面，而如果页面的样式选择器与 app.wxss 全局样式相同，则会覆盖全局样式。

理解了稍微复杂的小程序全局配置，接下来重点讲解 pages 目录下的文件（如图 3.4 所示）。

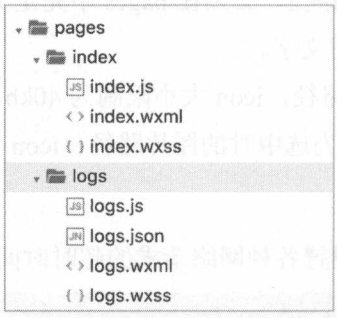


图 3.4

从图 3.4 中可以比较清晰地看出，一个小程序页面由四个文件组成，即用来编写 Javascript 的 js 文件、用来配置页面窗口表现的 json 文件、用来构建页面的 wxml 文件以及用来定义样式的 wxss 文件。这四个文件具有相同的文件名并放置在同名的目录下，该组织结构是固定的，并且这四个文件无须显式的相互引用，小程序会自动通过文件名找到相应文件。

在 js 文件中，小程序会默认自动调用 Page()方法，其接受一个 object 参数，该参数实现了如下事件作为默认调用。

- onLoad：生命周期函数，页面加载时调用，只会被调用一次。

- onReady：生命周期函数，页面初次渲染完成时调用，只会被调用一次。
- onShow：生命周期函数，页面显示时调用，可能会被多次调用。
- onHide：生命周期函数，页面隐藏时调用，可能会被多次调用。
- onUnload：生命周期函数，页面卸载时调用，只会被调用一次。

以上几个事件实现了页面的生命周期，除此之外，还实现了如下默认调用的事件和属性。

- onPullDownRefreash：监听用户下拉动作。
- data：用于页面的数据绑定。
- onReachBottom：监听用户上拉触底动作，顾名思义，用户将页面上拉至底部就会触发。
- onShareAppMessage：用户在小程序页面点击右上角分享时触发。

示例代码：

```
//index.js
Page({
 //用于实现页面的数据绑定
  data: {
    text: "This is page data."
  },
  onLoad: function(options) {
    // 当页面加载时调用，只会被调用一次
  },
  onReady: function() {
    // 页面初次渲染完成时调用，只会被调用一次
  },
  onShow: function() {
    // 页面显示时调用，可能被多次调用
  },
  onHide: function() {
    // 页面隐藏时调用，可能被多次调用
  },
  onUnload: function() {
    //  页面卸载时调用，只会被调用一次
  },
```

```
  onPullDownRefresh: function() {
    // 用户进行下拉动作时调用，需要在 config 的 window 选项中开启
enablePullDownRefresh，并且当处理完数据刷新后，wx.stopPullDownRefresh 可以
停止当前页面的下拉刷新。
  },
  // 事件方法.
  viewTap: function() {
    this.setData({
      text: 'Set some data for updating view.'
    })
  }
})
```

WXML 文件即前文提到的 WeiXin Markup Language，是微信小程序框架定义的一套标签语言规范，用于构建页面的结构。

在 WXML 中，可以完成数据绑定、列表渲染、条件渲染、模板以及事件绑定的任务，这里简单介绍数据与事件绑定，其他的详细用法将在后续章节里详细介绍。

数据绑定

以下内容通常写在 index.wxml 文件中：

```
<!--wxml-->
<view> {{message}} </view>
```

以下内容通常写在 indexjs 文件中：

```
Page({
  data: {
    message: 'Hello MINA!'
  }
})
```

以上代码完成了通过 wxml 中的{{mesage}}，与 js 代码中的 data.message 实现数据绑定，程序运行时会在界面中打印出：Hello MINA!

事件绑定

事件绑定的方法也很简单，在 wxml 文件中添加如下代码：

```
<view bindtap="add"> {{count}} </view>
并在 js 文件中添加如下代码:
Page({
  data: {
    count: 1
  },
  add: function(e) {
    this.setData({
      count: this.data.count + 1
    })
  }
})
```

即实现了在 view 标签上通过 bindtap 属性绑定了 add 触摸事件，改变绑定的数据内容需要通过 this.setData 方法，有关事件的用法会在后续章节详细讲解。

3.3　微信 Web 开发者工具使用方法介绍

为了让读者能够快速上手小程序开发，下面简单介绍下微信开发工具的使用方法。

微信 web 开发者工具是微信提供给用户用来开发小程序和公众号的一个工具，官网下载地址为：

https://mp.weixin.qq.com/debug/wxadoc/dev/devtools/download.html?t=20161107

微信的开发工具总体而言比较好用，代码提示也做的比较好，对于小程序开发，能较好地提高效率。

打开任意一个小程序项目，微信开发者工具的界面如图 3.5 所示。

打开小程序项目后，默认在编辑界面，可以看到左侧状态条中“编辑”按钮被选中，在此界面中，可以看到编号为（1）所示的区域为示图区，官方文档中称之为“模拟器”，可以在这里实时看到小程序的运行效果；（2）为小程序项目的目录结构，这里的详细用法在本章手把手环节以及后续章节还会详细讲到；（3）为代码编写窗口，可以在这里编写相应代码，代码编辑器具有自动保存、代码自动补全等特性；（4）为不同开发模式的切换按钮，以及下半部分的一些功能操作。接下来我们依次简单介绍。

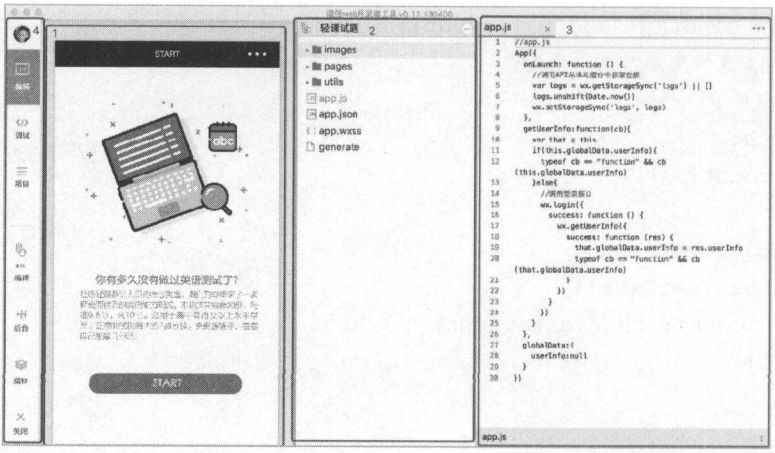


图 3.5

（1）编辑界面如图 3.5 所示，在编辑模式下，主要的工作即代码的编写，在该界面中，开发工具也提供了小程序“模拟器”，我们写完代码即可看到代码在小程序“模拟器”中呈现的效果，而“模拟器”呈现的效果与最终的实际效果基本相同。

（2）调试界面如图 3.6 所示，在调试模式下，我们主要用来进行代码的调试。该界面的操作大部分继承了 chrome 的调试界面，但是有几个比较特殊的面板，由于小程序没有 dom 的概念，对应地，也就没有了 chrome 的 Element 面板，取而代之的是 Wxml 面板，因此，如果需要调试小程序页面的结构与样式，就需要在 Wxml 面板中进行，这一点尤其需要注意。

图 3.6

小程序调试模式下另一个比较特别的面板为 Sensor，如图 3.7 所示，该面板供我们在“模拟器”中模拟传感器的情况，配合调试重力感应 API 与罗盘 API。

图 3.7

（3）项目界面如图 3.8 所示，项目面板的主要功能是对小程序进行实机调试，以及完成对小程序项目的一些配置，在配置信息选项页，可以查看我们在管理后台配置的合法域名情况。

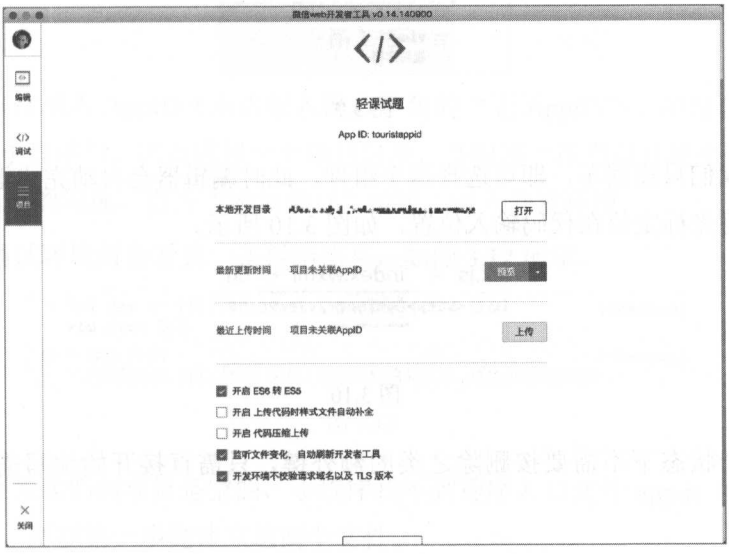


图 3.8

图 3.8 所示中由于是未关联 AppID，因此未显示配置选项标签，如果在创建项目时，关联了 AppID，则此界面还会显示配置选项标签。

另外进入项目面板时，我们会发现“编译”、“后台”、“缓存”这三个选项按钮不见了。说明这三个操作对于项目面板是不适用的。

（4）编译就对现有代码进行重新编译，在编写的代码因为种种因素未能立即生效，或者我们怀疑代码没有生效时，可以通过该按钮强制开发工具对代码进行重新编译。

（5）后台，用于模拟将小程序置于后台的情况，可以调试 onShow 等周期函数以及后台音乐播放等情况。

（6）缓存功能可以用来清除存储的小程序缓存数据。

（7）关闭功能即关闭开发者工具。

有一个小技巧需要注意，在编辑模式下编写 wxml 代码时，不要直接输入“<”符号开始写组件，否则写选择了代码提示的组件后语法可能会有问题，还需要进一步调整。而是应该直接输入组件名称，如需要 view 组件，就直接从 v 开始输入，输入到 vie 时，编辑器已经能推测出我们需要的是 view 组件，并且已经在提示列表中默认选中了 view 组件，如图 3.9 所示。

图 3.9

此时我们只需回车，即可选择这个组件，此时编辑器会自动完成组件代码的补充，并将光标定位在代码输入位置，如图 3.10 所示。

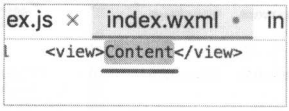


图 3.10

在这个状态下不需要按删除之类的额外键，只需直接开始编写需要的代码即可。

3.4　手把手教你做 Demo——Hello World 小程序

下面通过一个案例来讲解如何利用前面讲解的内容来进行实战。

3.4.1　Demo 的简要开发步骤

我们通过从无到有创建一个项目来熟悉下本章内容。

（1）打开“微信 web 开发者工具”，在弹出的对话框中选择“添加项目”按钮，如图 3.11 所示。

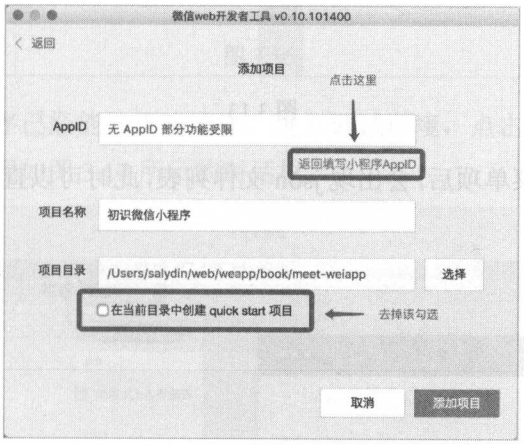


图 3.11

（2）依次填入 AppID（点击输入框右下角的“无 AppID”）、项目名称（例如“初识微信小程序”），以及选择一个项目目录，并取消“在当前目录中创建 quick start”复选框的勾选，点击“添加项目”按钮，进入开发环境。

进入调试环境后会看到一条报错信息，如图 3.12 所示。

图 3.12

这是因为我们的项目是空的，小程序找不到程序入口文件 app.js 以及配置文件 app.json，所以第一步先建立这两个文件。

（3）点击左侧“编辑”选项进入编辑环境，然后点击文件组织窗口右侧的三个点按钮，在弹出的菜单中点击“新建”选项，并从展开下级菜单中选择.json，如图 3.13 所示。

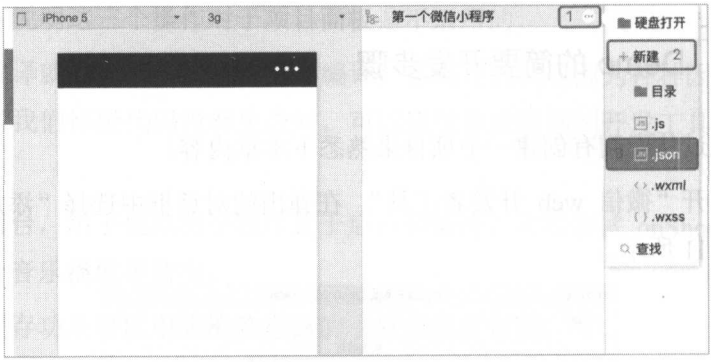


图 3.13

（4）点击.json 菜单项后，会出现 json 文件列表，此时可以直接输入文件名 app，如图 3.14 所示。

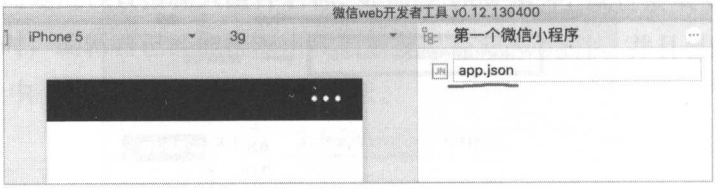


图 3.14

（5）这里先不编写 app.json 的内容，在编辑环境下点击左下侧的“编译”按钮，此时错误提示变为图 3.15 所示。

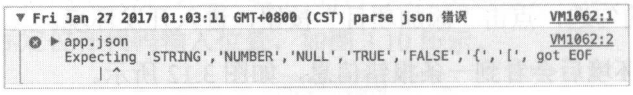


图 3.15

（6）这里先不理会这个错误，在 app.json 文件中编写如下代码：

```
{
  "pages":[
    "pages/index/index"
  ],
  "window": {
```

```
        "navigationBarTitleText": "初识小程序"
    }
}
```

（7）然后保存，此时会发现，小程序自动生成了所需的几个基本文件和目录，如图 3.16 所示。

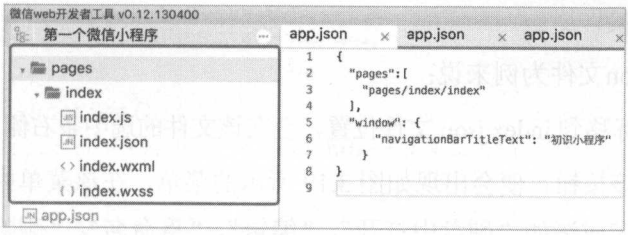


图 3.16

（8）此时小程序已经能够正确编译，进入调试环境，点击左侧工具栏的“编译”按钮，之后出现如图 3.17 所示的对话框。

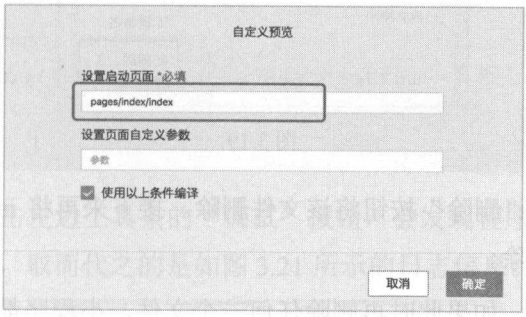


图 3.17

（9）按照图示，输入路径“pages/index/index”，然后点击“确定”按钮。即可运行成功，效果如图 3.18 所示。

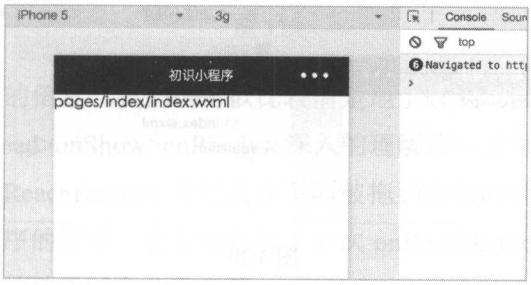


图 3.18

3.4.2 验证小程序可执行目录结构

可以看出，小程序不但生成了基本的文件及目录结构，也生成了一些基本的文件内容，并在页面中打印出了页面的路径，有兴趣的读者可以自行查阅代码。

为了验证小程序的最小可执行目录结构，这里依次删除 index.json 文件与 index.wxss 文件。

以index.json文件为例来说：

（1）将光标移到 index.json 文件位置，会在该文件的选中条右侧出现三点按钮。

（2）点击该按钮，便会出现如图 3.19 所示的菜单，在该菜单中，可以看见很多操作，可以完成诸如“硬盘中打开”、“编辑”、“重命名”、“删除”等操作，这里需要读者熟悉一下。

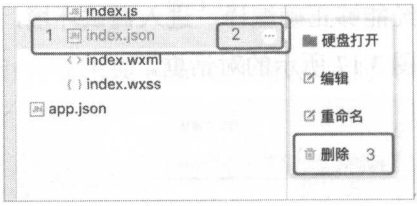


图 3.19

（3）这里点击“删除”按钮将该文件删除。接下来再将 index.wxss 文件按照同样的方法进行删除。

到此不难发现，如果此时再删除任何一个文件，小程序都会出现报错提示，因此，这几个最终的文件即是微信小程序最基本的结构要求，如图 3.20 所示。当然，这个程序除了显示了一个标题和页面路径以外没有任何功能，接下来，我们来依次实现一些本章提到的生命周期及数据绑定。

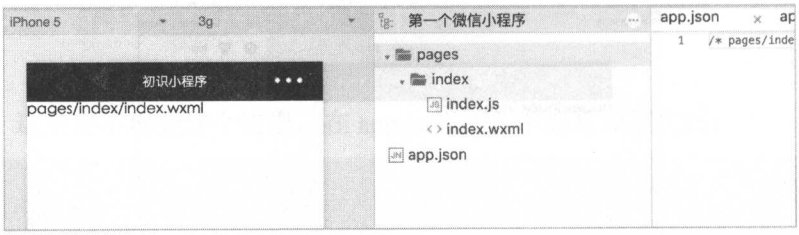


图 3.20

（1）打开 index.js 文件，会发现小程序框架已经自动生成了生命周期的方法

及详细的注释，我们将此文件的内容修改为如下代码。

```
Page({
  onLoad: function(options) {
    console.log('--index.js--OnLoad--页面加载')
  },
  onReady: function() {
    console.log('--index.js--onReady--页面初次渲染完成')
  },
  onShow: function() {
    console.log('--index.js--onShow--页面显示')
  },
  onHide: function() {
    console.log('--index.js--ohHide--页面隐藏')
  },
  onUnload: function() {
    console.log('--index.js--onUnload--页面卸载')
  },
  onReachBottom: function() {
    console.log('--index.js--onReachBottom--页面上拉触底')
  }
})
```

（2）保存后点击左边工具条的“调试”按钮，会发现程序正确运行，并且没有之前的警告信息，取而代之的是如图 3.21 所示的日志信息。

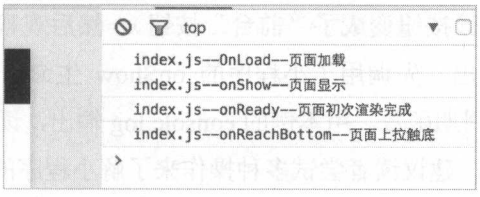


图 3.21

从控制台打印的信息，我们也可以比较清楚地了解到页面生命周期函数被调用的顺序，即 OnLoad>onShow>onReady，深入的理解这一点对今后的小程序开发有较大的帮助。onReachBottom 方法是在页面被拖到底部时调用，我们可以尝试往上拖动当前小程序的页面，会发现触发了多次 onReachBottom 方法，该方法在制作滚动列表加载的效果时会非常方便。

（3）接下来我们在项目根目录下创建一个 app.js 文件，并编写如下代码：

```
App({
  onLaunch: function() {
    console.log('app.js--onLanch--小程序初始化')
  },
  onShow: function() {
    console.log('app.js--onShow--小程序显示')
  },
  onHide: function() {
    console.log('app.js--onHide--小程序隐藏')
  }
})
```

保存后点击左边工具条的“调试”按钮，可以看到右侧的 Console 控制台打印出了相应的日志，从中我们很可以很清楚地看出，小程序，即写在 app.js 里的生命周期函数 App()是比写在页面里的生命周期函数 Page()的优先级更高，也就是说会先完成小程序即 App()里的生命周期函数，再调用页面即 Page()里的生命周期函数。

我们现在点击左侧工具条的“后台”按钮，观察右侧的 Console 控制台已经分别打印出了页面和小程序的 onHide 生命周期函数，并且顺序是先调用页面的 onHide 函数，再调用小程序的 onHide 函数，此过程与上述的生命周期即先小程序后页面的流程相反，接下来我们再点击左侧工具栏的“前台”按钮（点击了刚才的“后台”按钮后该按钮变成了“前台”按钮），然后观察右边 Console 控制台的日志信息，可以看出，先调用了小程序的 onShow 生命周期函数，然后打印了页面的 onShow 生命周期函数。用本例的 console.log 输出，读者可以很清楚地看到生命周期的运行情况，建议读者尝试多种操作来了解小程序的生命周期。

3.4.3 数据与事件的绑定

我们继续通过一个简单的例子来了解下基本的数据与事件的绑定。

（1）首先点击左侧工具栏“编辑”按钮进入编辑模式，在文件目录找到文件之前创建的 index.xml 文件，编写如下代码：

```
<view> {{ message }} </view>
```

该代码的{{}}即绑定代码的格式，内容为绑定的变量名，下一章还会详细介绍。

（2）然后在文件目录中打开 index.js 文件，在之前编写的 Page()方法中加入如下代码：

```
data: {
    message: 'Hello 小程序!',
  },
```

该方法用来配置初始化默认显示的数据，具体细节下一章节会讲。

（3）之后 index.js 文件中完整的代码内容为：

```
Page({
  data: {
    message: 'Hello 小程序!',
  },
  onLoad: function(options) {
    console.log('--index.js--OnLoad--页面加载')
  },
  onReady: function() {
    console.log('--index.js--onReady--页面初次渲染完成')
  },
  onShow: function() {
    console.log('--index.js--onShow--页面显示')
  },
  onHide: function() {
    console.log('--index.js--ohHide--页面隐藏')
  },
  onUnload: function() {
    console.log('--index.js--onUnload--页面卸载')
  },
  onReachBottom: function() {
    console.log('--index.js--onReachBottom--页面上拉触底')
  }
})
```

（4）保存该文件，点击左侧工具栏的“调试”按钮，即可看到小程序界面上显示了“Hello 小程序”字样，便实现了简单的数据绑定，在小程序中绑定事件更为简单。在 index.js 文件中继续加入如下代码：

```
tapName: function(event) {
console.log(event)
    this.setData({
      message:'好吧，我居然被点了',
    })
  }
```

该方法定义了用来被调用的 tapName 函数，并通过调用 setData 方法，改写了已绑定的 message 变量，以此实现更改界面上显示的文字内容。

但是要实现这一功能，我们还需要在 index.xml 文件中绑定这个事件，打开该文件，并在 view 标签中加入 bindtap="tapName"，其中 bindtap 属性即用来绑定点击事件，修改后的 index.xml 代码如下：

```
<view bindtap="tapName"> {{ message }} </view>
```

（5）保存后，点击左侧工具栏的“调试”按钮，进入调试模式，点击“Hello 小程序”字样，文字内容则会变为“好吧，我居然被点了”。至此，我们实现了数据及事件的绑定，以及通过绑定的事件修改原来绑定的数据内容，但目前的界面看起来有点乱，为了使界面显得整齐，我们继续实现一些简单的样式。通过前面所述的操作方法在 index 文件夹中创建一个 index.wxss 文件，并编写如下代码：

```
#v-msg{
    margin-top:80rpx;
    text-align: center;
}
```

可以看到其代码与 css 完全一样，只是其中的 rpx 单位比较特别，rpx 是小程序新增的一个尺寸单位，可以根据屏幕尺寸进行自适应，详细内容在下一章做介绍，这里大概了解即可。

（6）打开 index.xml 文件，并在 view 标签中加入 id=“v-msg”来指定样式，其语法和 html 里完全一样，完成修改的 index.xml 文件完整的代码为：

```
<view id="v-msg" bindtap="tapName"> {{ message }} </view>
```

（7）保存该文件，实际上就已经可以在界面中看到了居中和上边距的效果了，可见小程序的样式在编辑模式中是所见即所得的，十分方便。甚至还可以直接在编辑模式点击“Hello 小程序”来实现事件绑定功能的预览。完成后的效果如图 3.22 所示。

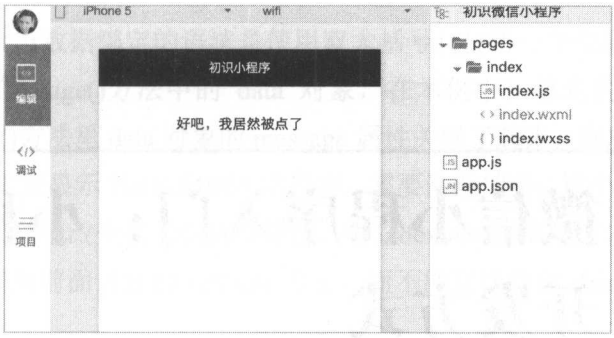


图 3.22

如果需要回到初始页面，可以点击左边工具栏的“编译”按钮。

3.5　本章要点总结

本章通过一个非常简单的实例初步了解了小程序的基本结构以及开发的基本步骤。

小程序拥有自己的生命周期，分为全局的小程序生命周期，以及页面的生命周期，通过手把手实例部分做了比较详细的介绍，其对于后续的开发非常关键，希望读者多实践本章实例，做到熟练掌握。

对于小程序的结构，可以简单地总结为全局的三个文件和页面的三个文件；全局的三个文件分别为 app.js、app.json 和 app. wxss，顾名思义，这三个文件都与小程序全局的功能有关，app.js 用来实现全局的小程序生命周期，以及一些需要全局处理的代码；app.json 则是进行一些小程序的全局配置，而 app.wxss 则是进行全局的样式编排，而页面的 wxss 文件的样式会覆盖全局的 app.wxss 文件的同名样式。页面的三个文件都放在一个同名的目录下组成一个页面，这三个文件名也必须相同，由此带来的好处是不需要相互引用，小程序会自动完成，.js 文件用来编写 javascript，wmxl 的功能与 html 相同，但标签与 html 完全不一样，.wxss 则是担任了 css 样式表的角色，只是在此基础上增加了一些小程序额外的扩展。

实际上小程序总的开发模式及思路基本如此，有了本章基础，读者可以按照自己的想法尝试多改动本章实例，甚至是自行编写一些简单的应用，如果能带着实际开发中的一些小问题进入下一章，相信会有更大的收获。

第 4 章

微信小程序入门：小程序的开发方式

上一章对小程序做了一个初步的了解，本章继续介绍小程序的数据绑定、事件、样式及几个关键的组件。

4.1 WXML 及其数据绑定

WXML 是微信小程序框架的标记语言，它的语法结构与 HTML 完全一致，但是其原有标签在 WXML 中已经彻底不适用，取而代之是 WXML 自己的组件标签，并扩展了数据绑定、模板的功能，甚至可以做模块化的引用。

上一章已经接触了简单的数据绑定：

index.wxml 文件中的代码（为了便于理解，省略了文件中的其他无关的代码）：

```
<view> {{ message }} </view>
index.js 文件中的代码
Page({
  data: {
    message: 'Hello 小程序!'
  }
})
```

从中不难看出数据绑定的语法是使用双大括号{{}}将一个变量括起来，该变量的内容则来自 Page()方法中的 data 对象，在本例中，绑定的{{message}}与 index.js 中 Page()方法里 data 对象的 message 属性关联了起来，程序运行后，便会在{{message}}位置显示 Hello MINA 字符串。需要注意的是，这个关联是持久的，即之后如果在代码中改变已关联的变量值，则 index.wmxl 中的内容也会相应的改变，但是需要使用前面讲过的 setData 方法，而不能直接修改 data 里的属性（即变量）值。

除了直接绑定标签的完整内容，WXML 也可以绑定标签属性的一部分，比如需要动态的改变前面讲过的 View 标签的 id 内容，可以在 index.wxml 文件中这样写：

```
<view id="v-msg{{id}}" ></view>
然后在 index.js 中加入这样的绑定代码：
Page({
  data: {
    id: 0
  }
})
```

在运行后这个 view 标签的 id 会被渲染为 v-msg0，但是在本例中这样写会影响样式文件的引用，使得之前写的居中样式失效，这里只需做一个小的改动即可恢复样式的引用，即把 id:0 改为空字符串 id:“”即可，由于绑定的是空字符串，v-msg{{id}}属性解析的最终结果仍然是 v-msg，因此又恢复了对样式的引用，即恢复了居中样式，读者可以动手尝试一下。

WXML 的数据绑定支持条件控制，一个简单的条件控制的语法如下：

```
<view wx:if="{{condition}}"> </view>
```

其中的 xf:if 为条件控制的属性，其后绑定的{{condition}}值如果为真，则显示 view，否则不显示，可以将例子里的 index.wxml 文件改写为：

```
<view id="v-msg" wx:if{{isshow}}></view>
```

然后在 index.js 中的 Page()方法中增加如下代码：

```
isshow:true
```

注意不要用引号引 true。

再运行程序，一时看不到什么变化，但是当把 index.js 文件中的 isshow:true 的值改为 false，再保存运行，会发现小程序界面中的“Hello 小程序”字样已经消失，实际上这个标签在最终也不会被渲染在页面上。运行程序后可以通过点击左侧工具栏的“调试”按钮进入调试模式，切换编辑模式与调试模式的方法相信读者已经掌握，后续不再赘述。然后在右侧控制台点击“Wxml”标签，来查看页面输出的标签内容，如图 4.1 所示。

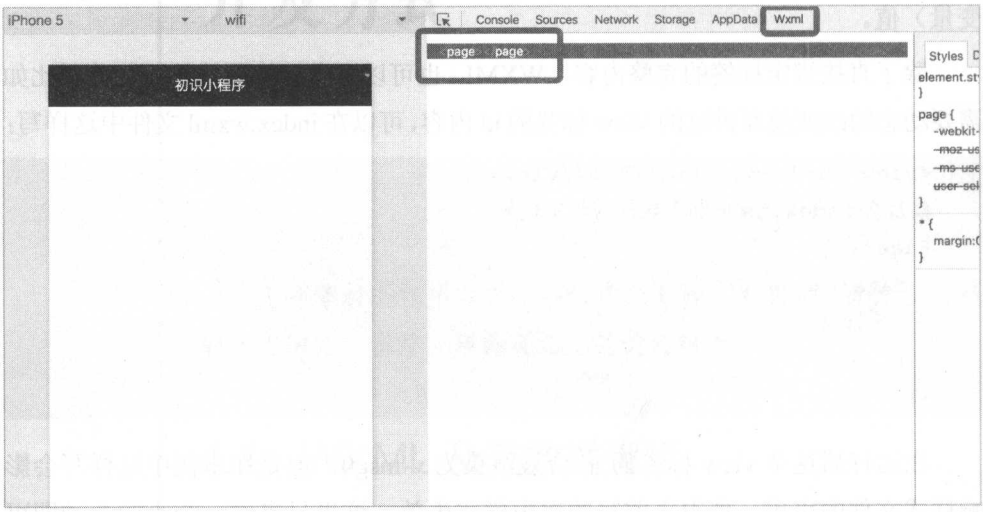


图 4.1

如果没有找到 WXML 的标签，也可能隐藏在右侧菜单里，点击该按钮即可在弹出的菜单里看到 WXML 项，点击之后便会在调试选项里出现。

从图中可以很容易看到，通过 wx:if={{isshow}}控制的 view 标签，在 isshow 被置为 false 后，就完全没有在最终的 WXML 中进行渲染，这一点需要读者注意。另外如果我们不是通过绑定来控制标签隐藏，而是直接写在 wx:if= 中，则需要注意千万不要写成 wx:if=“false”，因为“false”会被 WXML 解析为字符串，而字符串的值为真，因此 wx:if=“false”控制的标签会显示在小程序页面上，因此应该写成 wx:if=“{{false}}”，读者最好亲自手动尝试一下，以便加深记忆。

WXML 的 wx:if 属性还支持条件判断，将上例中的 wx:if={{isshow}}改为 wx:if={{isshow>5}}，然后再修改 index.js 中 Page()方法里的 isshow 值为 4，会发现界面中的文字没有显示，再改为 6，界面中的文字被再次显示出来，说明 isshow>5 的判断条件已生效。

除此之外 wx:if 属性还支持 elseif 与 else 分支，只不过 elseif 的表达在 WXML 中是 wx:elif，这里和我们直观的 elseif 还有些区别，else 分支对应的属性为 wx:else，我们将 index.wxml 的内容修改为如下代码：

```
    <view id="v-msg{{id}}"> isshow={{isshow}} </view>
    <view id="v-msg{{id}}"  wx:if="{{isshow<=10}}"> 小于或等于 10</view>
    <view id="v-msg{{id}}"  wx:elif="{{isshow>10&&isshow<=50}}"> 大于
10,并且小于或等于 50 </view>
    <view id="v-msg{{id}}"  wx:else> 大于 50 </view>
```

然后按照条件区间，修改 index.js 中 Page()方法里 isshow 的值，可以看到页面上相应的变化，如图 4.2 所示。读者可以多尝试更改条件区间，来体会一下条件语句的用法。

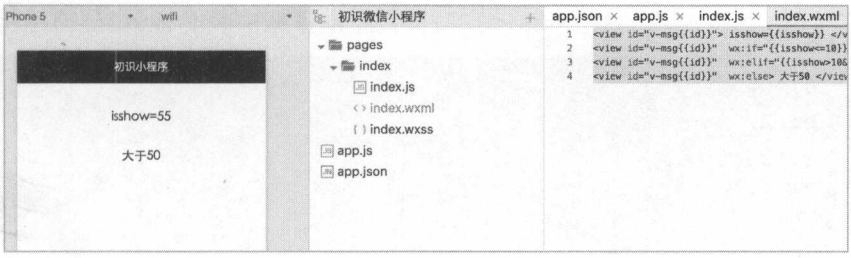


图 4.2

这里通过在每一个 view 里添加 wx:if 属性来实现了根据条件来进行单个标签的显示与否，如果需要对一组标签进行控制，WXML 也提供了<block/>标签来实现，<block/>不是组件，只是一个组件容器，用来对一组组件进行控制，但其本身不会在页面上做任何渲染。可以通过将 index.xml 文件的内容修改为如下代码来实际体验一下：

```
    <block wx:if="{{false}}">
    <view id="v-msg{{id}}"> isshow={{isshow}} </view>
    <view id="v-msg{{id}}"  wx:if="{{isshow<=10}}"> 小于或等于 10</view>
    <view id="v-msg{{id}}"  wx:elif="{{isshow>10&&isshow<=50}}"> 大于
10,并且小于或等于 50 </view>
    <view id="v-msg{{id}}"  wx:else> 大于 50 </view>
    </block>
```

可以看到在<block/>标签内的所有组件都因为设置了 wx:if="{{false}}"属性而不可见了，最终也不会在页面上做任何渲染。

WXML 还支持在{{}}内进行运算，包括三元运算、算数运算、逻辑判断、字符串运算和路径运算，下面分别做一下阐述。

三元运算即我们通常所指的问号运算符，在 JavaScript 里面也经常会用到，在 WXML 里可以直接使用三元运算，举个简单的例子。

```
<view wx:if="{{isshow?true:false}}">
```

在 WXML 里使用算数运算符需要注意只有{{}}里的运算符才会参与运算，其他的运算符则原样输出，参照以下的例子：

在 WXML 文件中编写如下代码：

```
<view> {{a + b}} + c </view>
在 js 文件中编写如下代码：
Page({
  data: {
    a: 1,
    b: 2,
    c: 3
  }
})
```

则在页面上输出的内容为：

```
3+c
```

字符串运算则非常简单，通过一个简单的例子熟悉一下。

在 WXML 文件中编写如下代码：

```
<view>{{"hello"+text}}</view>
```

然后在 js 文件中编写如下代码：

```
Page({
  data:{
    text: '小程序'
  }
})
```

则会在页面中输出如下文字：

```
hello 小程序
```

在 index.wxml 文件中编写如下代码：

```
<view class="contain">
    <!--定义一个模板-->
    <template name="myTemplate">
        <view>
            <text> for: {{for}} </text>
            <view><!--一个小技巧，用它来换行--></view>
            <text> bar: {{bar}} </text>
        </view>
    </template>
    <!--引用定义好的模板-->
    <template is="myTemplate" data="{{for: a, bar: b}}"></template>
</view>
```

然后在 index.js 文件中编写如下代码：

```
Page({
  data:{
      a:1,
      b:2
  }
})
```

这里在 index.wxml 文件中定义了一个模板，我们接下来的内容中就会详细介绍，这里读者先按照 wxml 文件中的注释先简单了解下，简单来讲，我们用<template name="myTemplate"> 标签定义了一个模板，然后接着用 <template is="myTemplate" data="{{for: a, bar: b}}">标签引用了这个模板，并通过 data 属性向引用的模板 myTemplate 传入了绑定数据，因此在模板定义中，就可以通过{{for}}来直接访问传入的绑定数据，而这里的对象 a,和 b 的值，则是通过在 js 中的 data 对象中绑定了其值 1 和 2。而最终既解析为了模板引用后的值。

这一块读者可以通过该代码自行练习几次就能很好的理解了。

至于其他的数据路径运算、组合、数组也都是通过类似的方法在标签的 data 属性中直接处理对象、数组等，包括比较特殊的扩展运算符...，也是实现在 data 属性中展开一个对象。

并且这几个方法多用于 template 标签的 data 属性中。

列表渲染是 WXML 提供的一项比较强大的功能，可以很方便地将 json 数组

渲染为列表项。只需在组件里的 wx:for 属性来绑定需要遍历的数组，即可将绑定的数组内容遍历展示出来。

我们在本例中的 index.js 文件的 Page()方法中增加如下代码：

```
colorArray: [{value: 'red'},
             {value: 'yellow'},
             {value:'black'},
             {value:'white'}]
```

然后在 index.wxml 文件中增加如下代码：

```
<view wx:for="{{colorArray}}">
  {{index}}: {{item.value}}
</view>
```

运行改代码，会看到如图 4.3 所示的显示效果。

图 4.3

其中第一列 0、1、2、3 为我们在 index.js 中 Page()方法里写的 colorArray 数组的下标，数组一共有四个元素，每个元素都是一个 json 对象，数组下标从 0 开始，可以用默认的{{index}}来指定，数组当前元素的默认变量名为 item，通过{{item.<下标变量名>}}来指定，也可以通过 wx:for-index 属性来指定数组下标的名称，用 wx:for-item 属性来指定当前元素的变量名，例如讲上述例子中 index.wxml 代码修改为：

```
<view wx:for="{{colorArray}}" wx:for-index="index" wx:for-item=
"item">
  {{index}}: {{item.value}}
</view>
```

因为其指定的 wx:for-index 数组下标名称与 wx:for-item 当前元素变量名与默认相同，所以不用修改其标签内数据绑定的内容，但如果将 wx:for-index 的值修改为 order，wx:for-item 的值修改为 key，则需要相应的修改该标签内数据绑定的内容为：{{order}}: {{key.value}}，完整的 wxml 代码内容为：

```
<view wx:for="{{colorArray}}" wx:for-index="order" wx:for-item=
"key">
  {{order}}: {{key.value}}
</view>
```

wx:for 属性也可以嵌套，官方提供了一个很直观的九九乘法表：

```
<view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for-item="i">
  <view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for-item="j">
    <view wx:if="{{i <= j}}">
      {{i}} * {{j}} = {{i * j}}
    </view>
  </view>
</view>
```

将上述代码直接写入 wxml 文件并进入调试模式即可看到运行效果，其做了两层嵌套，并在里层做了大小判断以控制输出，用 javascript 写过 for 循环的用户应该很容易理解这段代码，该经典编程题巧妙地运用了 wxml 的循环，控制判断及表达式运算。

在实际开发中，我们通常会循环输出一整块包含多个组件的内容，与 wx:if 一样，可以使用<block wx:for="{{colorArray}}">来表示如下。

```
<block wx:for="{{colorArray}}">
<view> 索引:{{index}} </view>
<view> 值:{{item.value}} </view>
</block>
```

在大多数开发实践中，我们经常会使用多类组件组合成单个列表项，再根据数组循环输出，并会在后续动态的加入或者删除列表项，而对于包含诸如输入文本框<input/>以及开关选择器<switch/>等组件而言，我们在通过绑定的数组内容来实现列表项对应变化时，可能需要保持用户的开关选择、input 输入等状态不发生改变，此时则可以使用 wx:key 来指定一个数组里的唯一标识符字段，而如果循环输出的 item 本身就是一个唯一的字符串或数字，也可以指定保留关键字*this，在动态改变列表内容时指定 wx:key 属性，框架会根据指定的 wx:key 对列表项重新排序，而不是重新渲染，以此提高性能，具体用法在本章后续案例结合事件的实现再做详细的介绍。

WXML 还直接支持模板，可以将页面中的导航和底部工具栏等公用区域做成

模板以供在其他页面调用，从而实现包括页面布局及样式在内的代码复用。

创建模板的标签为<template name=“”></template>，其中的 name 属性用来指定该模板的名称，而使用模板时使用的也是<template/>标签，只是需要用 is 属性指定要使用的模板名（即创建模板时用 name 指定的名称）。

在 wxml 文件中编写如下代码：

```
<template name="nav">
    <view>这里是导航区域</view>
</template>
```

以上代码创建了名为 nav 的模板，但是运行这段代码并不会在页面中有任何输出，因为只是创建了模板，并未使用模板。

接下来在同一文件中加入如下代码：

```
<template is="nav">
</template>
```

保存并进入调试模式运行代码，会看到页面中显示了“这里是导航区域”字样，说明模板已经被调用。

也可以在创建的模板中定义绑定的数据，如：

```
<template name="nav">
    <view>{{text}} </view>
</template>
```

并在 js 文件中的 Page()函数的 data 对象中加入 text:"这里是导航"项。

然后在使用该模板时，用 data 指定绑定数据的内容，如代码所示：

```
<template is="nav" data="{{text}}">
</template>
```

其执行效果与上例一致。

is 属性也可以结合{{}}语法（即官方文档所说的 Mustache 语法），动态的指定需要使用的模板名，此处可以结合数据绑定支持的所有特性来使用不同模板，例如如下代码。

```
<template name="jishu">
  <view> 奇数 </view>
</template>
```

```
<template name="oushu">
  <view> 偶数 </view>
</template>

<block wx:for="{{[6,7, 8, 9, 10]}}">
    <template is="{{item % 2 == 0 ? 'oushu' : ' jishu'}}"/>
</block>
```

通过使用三元操作符，实现了判断遍历的数组项是否能被 2 整除，如果可以整除说明该项为偶数，则使用名为 oushu 的模板，如果不可以整除说明该项为奇数，则调用名为 jishu 的模板，运行后会打印出：

```
偶数
奇数
偶数
奇数
偶数
```

与数组[6,7, 8, 9, 10]项相吻合。

4.2　WXSS——小程序的 CSS 样式

WXSS 在微信小程序中的定义是用于描述 WXML 组件样式的一套语言，全称为：WeiXin Style Sheets。通俗点说，它就是小程序的 CSS，它对 CSS 的大部分标准做了支持，并在此基础上做了一点扩展，但是其语法和 CSS 语法是完全一致的，即完全可以直接使用 CSS 语法来编写 WXSS 语言。但是对于 WXSS 扩展的部分，我们也有必要做一下了解。

4.2.1　新的尺寸单位 rpx

小程序提供了一个新的尺寸单位 rpx，可以根据屏幕宽度进行自适应。这样在开发的时候，就可以不用在屏幕自适应方面花费太多精力，我们也建议在小程序开发中，优先考虑使用 rpx 单位，当然，也并不是说所有的单位都要生硬的使用 rpx，因为也有一些场景下，并不一定所有的元素都需要完全的自适应，有一些元素是需要固定大小的，此时就更适合使用 px 控制。

小程序规定屏幕宽为 750rpx，换算关系以 iPhone 6 为例，屏幕宽度为 375px，共有 750 个物理像素，则 750rpx = 375px = 750 物理像素（1rpx = 0.5px = 1 物理像素）。如果不理解这个换算关系，也并无大碍，只需参照官方文档，提供的参数来使用该单位即可，如表 4.1 所示。

表 4.1　设备尺寸换算

设备	rpx 换算 px（屏幕宽度/750）	px 换算 rpx (750/屏幕宽度)
iPhone 5	1rpx = 0.42px	1px = 2.34rpx
iPhone 6	1rpx = 0.5px	1px = 2rpx
iPhone 6 Plus	1rpx = 0.552px	1px = 1.81rpx

因为 iPhone 7 与 iPhone 7 Plus 和上面的换算系统类似，这儿不再详细说明。

通过上表可以很容易的推导出，1rpx =（屏幕实际宽度/750）px，以 iPhone 6 Plus 为例，屏幕的宽度为 414，带入这个公式即 1rpx =（414/750）px = 0.552px，在设计图纸时，官方文档建议使用 iPhone 6 的尺寸作为设计稿的标准，但我们建议以 iPhone 6 尺寸的两倍为设计稿标准，这样一方面我们可以看到更多细节，方便设计标记尺寸，一些 UI 元素也可以避免缩小时失真产生毛刺。

4.2.2　样式导入

微信 WXSS 还原生支持使用@import 导入样式，方便我们对公共的样式进行复用，这一点在现有很多流行 CSS 库上都很常见。

假如有一个 common.wxss 文件，其中有一些公用的样式，如：

```
.nav {
  padding:5px;
}
```

然后在同目录下有一个 index.wxss 文件，可以用如下方法导入。

```
@import "common.wxss"

.content{
  margin:5px;
}
```

则 index.wxss 里的代码就相当于：

```
.nav {
  padding:5px;
}

.content{
  margin:5px;
}
```

因此，在导入样式文件的时候，读者也要多留意样式覆盖的问题。

4.2.3　内联样式

微信 WXSS 支持内联样式，写法与在 HTML 里一样，即：

```
<view style="padding:5rpx;" > </view>
```

除此之外，WXSS 还支持数据绑定，可以实现通过绑定的样式来控制样式，如：

```
<view style="padding:{{padding}};" />
```

并且官方文档建议使用数据绑定的方式来控制组件样式，因为 style 接收动态的样式，在运行时会进行解析，可以提高一些渲染的速度，所以建议读者在开发中可以多加留意。

另外 WXSS 也可以和 HTML 一样指定样式名：

```
<view class="nav-color content-font" />
```

多个样式类名之间用空格隔开，其用法和在 HTML 里是一样的，只是写在 WXML 的组件里，这里就不再赘述。

4.2.4　选择器

目前 WXSS 支持的选择器支持如官方文档提供的表 4.2 所示。

表 4.2　WXSS 支持的选择器列表

选 择 器	样　例	样例描述
.class	.intro	选择所有拥有 class="intro" 的组件
#id	#firstname	选择拥有 id="firstname" 的组件

续表

选 择 器	样　例	样例描述
element	view	选择所有 view 组件
element, element	view checkbox	选择所有文档的 view 组件和所有的 checkbox 组件
::after	view::after	在 view 组件后边插入内容
::before	view::before	在 view 组件前边插入内容

对于小程序开发，这几个选择器已经足够我们使用了。

4.3 事件

有一定 javascipt 基础的用户对事件一定不会陌生，简单来说，事件是用来响应用户某个操作的方法调用，在小程序中，使用事件的方法就是在 WXML 组件中，通过相应的事件属性指定一个方法名，如我们已经熟悉的 bindtap 属性，指定的方法名与 js 文件里 Page()函数中需要绑定的函数同名，即完成了事件的绑定，当该事件触发时（使用 bindtap 的触发条件为用户点击该组件时），便会调用在 Page()函数中绑定的函数。事件的使用在前面章节中已经有所接触，下面再通过一个简单的例子展开阐述。

在 index.wxml 文件中编写如下代码：

```
    <view id="tapEvent" data-hi="XiaoChengXu" bindtap="tapMe"> 快点我!
</view>
```

然后在 index.js 文件中编写如下代码：

```
    Page({
      tapMe: function(event) {
        console.log(event)
      }
    })
```

保存文件后切换到调试模式，点击页面中的文本区域“快点我”，可以看到右边的控制台打印出事件的 event 对象内容，如图 4.4 所示。

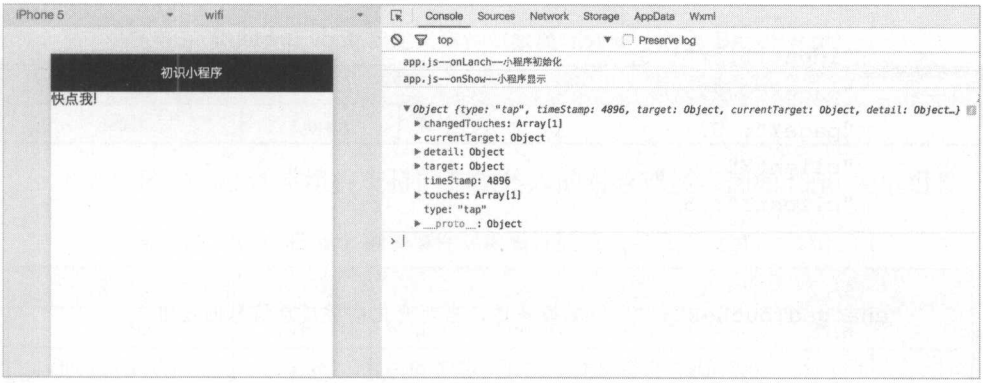


图 4.4

打印出来的内容大致如下：

```
{
  "type": "tap",  //事件类型，此处为 tap，即"手指触摸后马上离开"
  "timeStamp": 4327, //事件触发时的时间戳
  "target": {
    "id": "tapEvent", //事件源组件的 id
    "offsetLeft": 0,
    "offsetTop": 0,
    "dataset": {
      "hi": "XiaoChengXu" //事件源组件上由 data-开头的自定义属性组成的
集合。
    }
  },
  "currentTarget": {
    "id": "tapEvent",
    "offsetLeft": 0,
    "offsetTop": 0,
    "dataset": {
      "hi": "XiaoChengXu"
    }
  },
  "detail": { //组件额外的信息
    "x": 27,
    "y": 5
  },
  "touches": [ //触摸事件，当前停留在屏幕中的触摸点信息的数组
```

```
    {
      "identifier": 0,
      "pageX": 27,
      "pageY": 5,
      "clientX": 27,
      "clientY": 5
    }
  ],
  "changedTouches": [ //触摸事件，当前变化的触摸点信息的数组
    {
      "identifier": 0,
      "pageX": 27,
      "pageY": 5,
      "clientX": 27,
      "clientY": 5
    }
  ]
}
```

注意，为了便于理解，这里加了一些注释，但是在 json 格式里是不能加注释的，因为这是一种错误语法，如需要使用这些代码，请记得删去注释。

打印出来的这部分内容就属于事件对象，当组件触发事件时，除非组件有特殊说明，否则都会收到一个事件对象，以事件方法的参数传入。

微信官方文档提供了一些与事件对象相关的列表（见下表 4.3～4.8 所示），我们加以简单的解释。

表 4.3　BaseEvent 基础事件对象属性列表

属　性	类　型	说　明
type	String	事件类型
timeStamp	Integer	事件生成时的时间戳
target	Object	触发事件的组件的一些属性值集合
currentTarget	Object	当前组件的一些属性值集合

表 4.4　BaseEvent 表中 currentTarget 属性支持的参数

属　性	类　型	说　明
id	String	事件源组件的 id
tagName	String	当前组件的类型
dataset	Object	事件源组件上由 data-开头的自定义属性组成的集合

表 4.5　CustomEvent 自定义事件对象属性列表（继承 BaseEvent）

属　性	类　型	说　明
detail	Object	额外的信息

上表中的 Detail 通常组件支持的方法传入的对象属性，例如 form 表单组件。

表 4.6　TouchEvent 触摸事件对象属性列表（继承 BaseEvent）：

属　性	类　型	说　明
Touches	Array	触摸事件，当前停留在屏幕中的触摸点信息的数组
changedTouches	Array	触摸事件，当前变化的触摸点信息的数组

touches 是一个数组，每个元素为一个 Touch 对象（canvas 触摸事件中携带的 touches 是 CanvasTouch 数组）。表示当前停留在屏幕上的触摸点。

表 4.7　自定义事件对象属性列表

属　性	类　型	说　明
identifier	Number	触摸点的标识符
pageX, pageY	Number	距离文档左上角的距离，文档的左上角为原点，横向为 X 轴，纵向为 Y 轴
clientX, clientY	Number	距离页面可显示区域（屏幕除去导航条）左上角距离，横向为 X 轴，纵向为 Y 轴

表 4.8　自定义事件对象属性列表

属　性	类　型	说　明	特殊说明
identifier	Number	触摸点的标识符	
x, y	Number	距离 Canvas 左上角的距离，Canvas 的左上角为原点，横向为 X 轴，纵向为 Y 轴	

4.4　视图容器

至此，我们已经对小程序的基本语法有了了解，在此基础上，再来介绍几个在开发中比较基础的组件。学习开发语言基础是比较枯燥的，但是理解后，再了解组件和 API 的使用，就会感到轻松和有趣，通过运用组件实现一些功能后，也能为我们带来成就感，开发中最大的乐趣即是如此。为了能够更轻松地学习后续章节里组件和 API 的使用，希望读者再花一些时间快速巩固一下前两个章节里的内容。

4.4.1 view 视图容器

对于 view 组件我们已经不陌生了，它是一个视图容器，即可以把一些区域用 view 组件来分割，view 组件的基本使用方法如下。

```
<view> </view>
```

从使用上来讲，view 组件和在 HTML 中用到的 div 是完全一样的，甚至在开发中，也可以把往常需要使用 div 来实现的内容，在小程序中使用 view 组件来替换。但是在实现中，view 也不完全等价于 div，这些区别会在后续的开发中注意到，但是对开发来说，实际上是更加方便了。

在官方文档提供的实例中，使用到了 css3 的特性 flex 布局，这也说明 WXSS 支持 flex 布局，对于 flex 布局的使用，感兴趣的读者可以查阅下相关资料。

4.4.2 scroll-view 可滚动视图区域

在前端开发中，有一个 iscroll 组件库，用来实现在页面中的滚动区域，有一些前端开发经验的同学应该会对此比较熟悉，小程序的 scroll-view 组件像极了 iscroll 组件，在某些方面，其用法则比 iscroll 更为简单，功能上则更为强大。

scroll-view 组件的语法为：

```
<scroll-view></scroll-view>
```

根据 scroll-view 组件提供的属性和方法，可以实现很多有趣和实用的功能。

官方列举了 scroll-view 组件提供的所有功能，如表 4.9 所示。

表 4.9 scroll-view 组件的主要属性列表

属性名	类 型	默认值	说 明
scroll-x	Boolean	false	允许横向滚动
scroll-y	Boolean	false	允许纵向滚动
upper-threshold	Number	50	距顶部/左边多远时（单位 px），触发 scrolltoupper 事件
lower-threshold	Number	50	距底部/右边多远时（单位 px），触发 scrolltolower 事件
scroll-top	Number		设置竖向滚动条位置
scroll-left	Number		设置横向滚动条位置

续表

属性名	类　型	默认值	说　明
scroll-into-view	String		值应为某子元素 id，则滚动到该元素，元素顶部对齐滚动区域顶部
bindscrolltoupper	EventHandle		滚动到顶部/左边，会触发 scrolltoupper 事件
bindscrolltolower	EventHandle		滚动到底部/右边，会触发 scrolltolower 事件
bindscroll	EventHandle		滚动时触发，event.detail = {scrollLeft, scrollTop, scrollHeight, scrollWidth, deltaX, deltaY}

可见横向和纵向滚动默认都是禁止的，因此使用 scroll-view 组件实现区域滚动至少要有一个 scroll-x 或者 scroll-y 属性指定允许滚动的方向，如下面例子实现了横向滚动。

```
<scroll-view scroll-x="true"></scroll-view>
```

而如果需要实现竖向滚动时，则需要指定 scroll-view 组件的高度，否则 scroll-view 不知道在将操作限定在什么范围内，代码如下：

```
<scroll-view scroll-y="true" style="height: 100rpx;"></scroll-view>
```

upper-threshold 和 lower-threshold 两个属性分别与 bindscrolltoupper 和 bindscrolltolower 相对应，当滚动区域距 scroll-view 区域顶部或者左边的距离达到 upper-threshold 规定的范围内时，单位为 px，会触发 bindscrolltoupper 绑定的函数；当滚动区域距 scroll-view 区域底部或右边的距离达到 lower-threshold 规定的范围内时，则会触发 bindscrolltolower 绑定的事件函数，我们可以以区域的上半部分（顶部或左边）及下半部分（底部或右边）来记忆，upper-threshold 属性值会触发 bindscrolltoupper 事件，而 lower-threshold 属性值会触发 bindscrolltolower 事件。这两个事件对于开发滚动加载类的效果会非常的方便，不用自己去计算滚动的范围了。

scroll-top 和 scroll-left 用来设置滚动条的位置。

bindscroll 也是该组件比较关键的事件，会通过其事件对象中的 event.detail 传入滚动的坐标信息，便于我们开发更强大的功能。

下面通过一个简单的例子加深一下对该组件的理解。

为了便于理解，我们的例子都通过 index 目录来展示，读者也可以新建其他目录来尝试。

在 index.wxml 文件中编写如下代码：

```
<view>
    <scroll-view
        scroll-y="true"
        upper-threshold="20"
        bindscrolltoupper="scrolltoupperFun"
        lower-threshold="30"
        bindscrolltolower="scrolltolowerFun"
        scroll-into-view='v1'
        style="height:600rpx"
        >
        <view  id='v1'  style="height:300rpx;text-align:  center;
background-color: cadetblue;">
            这里是视图内容 1
        </view>
        <view  id='v2'  style="height:500rpx;text-align:  center;
background-color: darkkhaki;">
            这里是视图内容 2
        </view>
        <view  id='v3'  style="height:600rpx;text-align:  center;
background-color: indianred;">
            这里是视图内容 3
        </view>
    </scroll-view>
</view>
```

在 scroll-view 组件里又包含了三个 view 组件，每个 view 都有不同的 id 及 style 属性。

在 scroll-view 组件中，我们设置了 scroll-y 属性为 tue, upper-threshold 属性为 "20"，bindscrolltoupper 指定为"scrolltoupperFun"方法，之前已经详细介绍过，upper-threshold 设置了 20 后，会在左半边滚动接近 20px 范围内会触发 bindscrolltoupper 指定的方法，因为本例只设置了 scroll-y="true"，即竖向滚动，因此只有在滚动接近顶部 20px 范围内触发 bindscrolltouppersg 属性指定的 scrolltoupperFun 方法，属性 lower-threshold 和 bindscrolltolower 一样，是在滚动接近底部 20px 范围时触发 bindscrolltolower 属性指定的 scrolltolowerFun 方法。

scroll-into-view 属性是用来指定将 scroll-view 组件滚动到包含的子元素的顶

部，可以通过数据绑定实现将滚动位置滚动到指定的元素位置处，功能比较强大，实现起来也很简单，这里接着在 index.js 文件中编写如下代码：

```
Page({
  data: {
  },
  scrolltoupperFun:function(e){
      console.log('滚动到了顶部差 20 像素内，事件对象内容为:')
      console.log(e)
    },
  scrolltolowerFun:function(e){
    console.log('滚动到了底部差 30 像素内，事件对象的内容为:')
    console.log(e)
  }
})
```

在本方法中，我们定义了在 wxml 文件中指定的 scrolltoupperFun 和 scrolltolowerFun 两个方法，并在触发时打印了相应内容，并打印了事件对象 e，运行代码，可以看到如图 4.5 所示的效果。

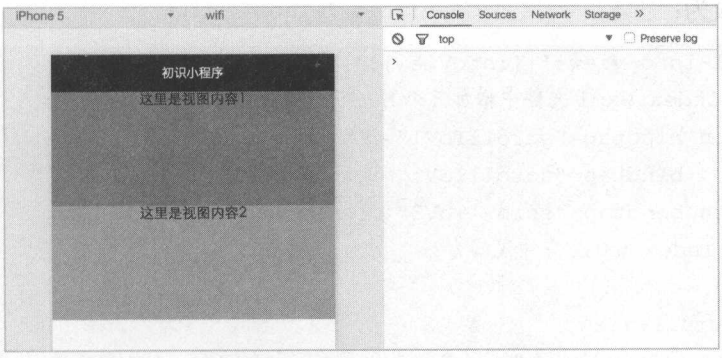


图 4.5

我们上下拖动视图区域，可以发现已经能拖动了，并在右侧出现滚动条，当把视图区域拖到底部位置时，会在控制台打印出输出的 log 信息，如图 4.6 所示。

图 4.6

可见用该方法检测是否拖动到底部非常方便。

现在我们改动代码，将 index.xwml 文件中 scroll-into-view 的属性从'v1'改为'v2'，在调试模式下点击“编译”按钮，运行效果如图 4.7 所示。

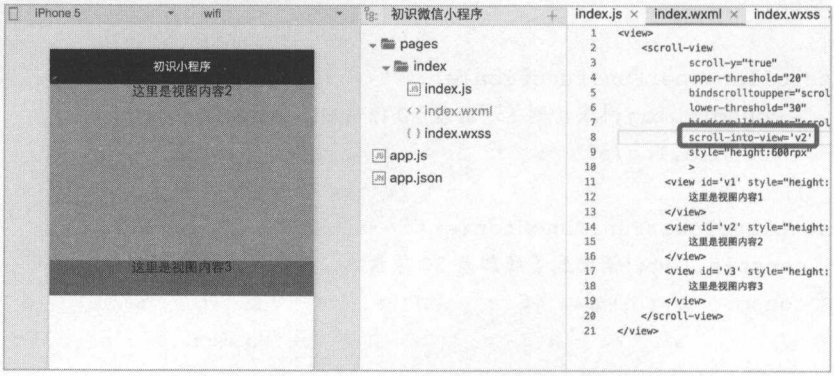


图 4.7

会发现已经滚动到 id 为 v2 的 view 组件顶部。

我们修改 index.wxml 文件中的 scroll-into-view 的属性值为：'{{intoView}}'，即代码修改为：

```
scroll-into-view='{{intoView}}'
然后在 index.wxml 文件中增加三个 button 组件，
<button bindtap="scrollToV1">滚动到 v1</button>
<button bindtap="scrollToV2">滚动到 v2</button>
<button bindtap="scrollToV3">滚动到 v3</button>
完整的 index.wxml 文件代码为：
<view>
    <scroll-view
            scroll-y="true"
            upper-threshold="20"
            bindscrolltoupper="scrolltoupperFun"
            lower-threshold="30"
            bindscrolltolower="scrolltolowerFun"
            scroll-into-view='{{intoView}}'
            style="height:600rpx"
            >
        <view  id='v1'  style="height:300rpx;text-align:  center;
background-color: cadetblue;">
```

```
            这里是视图内容 1
        </view>
        <view  id='v2'  style="height:500rpx;text-align:  center;
background-color: darkkhaki;">
            这里是视图内容 2
        </view>
        <view  id='v3'  style="height:600rpx;text-align:  center;
background-color: indianred;">
            这里是视图内容 3
        </view>
    </scroll-view>

    <button bindtap="scrollToV1">滚动到 v1</button>
    <button bindtap="scrollToV2">滚动到 v2</button>
    <button bindtap="scrollToV3">滚动到 v3</button>
</view>
```

然后在 index.js 文件中增加 data 初始数据绑定以及 scrollToV1、scrollToV2、scrollToV3 三个事件，完整的 index.js 文件内容为：

```
Page({
  data: {
    intoView:'v1'
  },
  scrolltoupperFun:function(e){
     console.log('滚动到了顶部差 20 像素内，事件对象内容为：')
     console.log(e)
  },
  scrolltolowerFun:function(e){
    console.log('滚动到了底部差 30 像素内，事件对象的内容为：')
    console.log(e)
  },
  scrollToV1:function(){
    this.setData({
     intoView:'v1'
    })
    console.log('滚动到 v1')

  },
  scrollToV2:function(){
```

```
    this.setData({
      intoView:'v2'
    })
    console.log('滚动到 v2')

  },
  scrollToV3:function(){
    this.setData({
      intoView:'v3'
    })
    console.log('滚动到 v3')

  }
})
```

该代码中，我们通过数据绑定方式，将 index.wxml 文件中的 scroll-into-view 值与 data 对象中的 intoView 进行了绑定，并设置了初始值 v1，然后通过 scrollToV1、scrollToV2、scrollToV3 事件，分别采用 this.setData()方法设置了 intoView 为对应的 v1、v2、v3，因此很简便地实现了对滚动视图滚动区域的控制，根据子组件内容的控制也是实际开发中最大的痛点，微信小程序的视图组件很巧妙地简化了这类需求的实现。

理解了本例，其他属性的使用就变得非常简单了，读者可以在本例基础上尝试实现其他属性和事件，强化对 scroll-view 组件的了解。

4.4.3 swiper 滑块视图容器

小程序里的滑块视图容器是用来实现“焦点图”效果的一个组件。“焦点图”的应用十分广泛，几乎所有知名互联网手机版网页都有“焦点图”区域，如图 4.8 所示。

可以看到无论是乐视、腾讯、网易还是淘宝，其首页都有焦点图功能，从而可见其应用的广泛性。而微信小程序的滑块视图容器封装了很常用的焦点图特性，使用起来也非常地方便。

图 4.8

滑块视图容器的标签为：

```
<swiper>  </swiper>
```

swiper 组件的主要属性可以参照官方提供列表，如表 4.10 所示。

表 4.10　swiper 组件的主要属性

属性名	类　型	默认值	说　明
indicator-dots	Boolean	false	是否显示面板指示点
autoplay	Boolean	false	是否自动切换
current	Number	0	当前所在页面的 index
interval	Number	5000	自动切换时间间隔
duration	Number	1000	滑动动画时长
bindchange	EventHandle		current 改变时会触发 change 事件，event.detail = {current: current}

可见其属性非常少，但是 swper 组件提供了使用 swiper-item 子组件来实现其灵活的展现形式，其宽高会自动设置为 100%。

下面来实现一个简单的焦点图，读者可以按照自己的需要，在项目中新建一个目录，在 pages 目录下新建一个 swiper 目录，然后在 swiper 目录里分别新建 swiper.js 和 swiper.wxml 文件，为了展现 swiper 目录里的页面，我们需要先修改一下 app.json 里"pages"数组增加"pages/swiper/swiper"项，并保证其放在首个，修改后的代码内容大致如下：

```
{
  "pages":[
```

```
      "pages/swiper/swiper",
      "pages/index/index"
    ],
    "window": {
      "navigationBarTitleText": "初识小程序"
    }
  }
```

，然后在 swiper.wxml 中编写如下代码：

```
<view>
    <swiper indicator-dots="true" style="height:500rpx">
        <swiper-item>
            <image
src="http://img02.tooopen.com/images/20150928/tooopen_sy_143912755726.jpg" width="355" height="150"/>
        </swiper-item>
        <swiper-item>
            <image
src="http://img06.tooopen.com/images/20160818/tooopen_sy_175866434296.jpg"  width="355" height="150"/>
        </swiper-item>
        <swiper-item>
            <image
src="http://img06.tooopen.com/images/20160818/tooopen_sy_175833047715.jpg" width="355" height="150"/>
        </swiper-item>
    </swiper>
</view>
```

保存后，可以在预览区看到效果，为了保证代码正确运行，这儿进入调试模式，运行后效果如图 4.9 所示。

可以看到，这里没有写任何的 js 代码，就已经实现了一个功能完备的焦点图。

在代码中，我们使用的 swiper-item 子标签包含了一个 image 标签，该标签的具体用法后面还会详细讲到，如果用户接触过 html 里的 img 标签，对此标签的用法也应该不会感到陌生，接下来在 swiper 组件中加入属性：autoplay="true"，interval="1000"，duration="100"，如图 4.10 所示。

图 4.9

```
swiper.wxml ×  app.json ×
1   <view>
2       <swiper indicator-dots="true"
3           autoplay="true"
4           interval="1000"
5           duration="100"
6       style="height:500rpx">
```

图 4.10

在编辑模式下可以看到焦点图开始以很快的速度自动切换，如果读者看着感觉头晕，也可以尝试自己修改这几个属性的值来优化，需要注意的是，interval 和 duration 属性值的单位是毫秒。

同样的思路，组件的属性都支持使用数据绑定的方式，来实现动态呈现和控制。我们加入一些数据和事件的绑定来加深对该组件的理解。

修改 swiper.xml 文件中 swiper 标签的属性 autoplay 的值为{{ isAutoplay}}，然后我们增加属性 bindchange 值为："picChange"。并在 swiper 标签之外增加一个 button 标签：

```
    <button bindtap="switchAutoplay">
        切换 autoplay 状态
</button>
```

完整的 swiper.xml 文件代码为：

```
<view>
    <swiper indicator-dots="true"
        autoplay="{{isAutoplay}}"
        interval="1000"
```

```
            duration="100"
            bindchange="picChange"
        style="height:500rpx">
            <swiper-item>
                <image
src="http://img02.tooopen.com/images/20150928/tooopen_sy_143912755726.jpg" width="355" height="150"/>
            </swiper-item>
            <swiper-item>
                <image
src="http://img06.tooopen.com/images/20160818/tooopen_sy_175866434296.jpg"  width="355" height="150"/>
            </swiper-item>
            <swiper-item>
                <image
src="http://img06.tooopen.com/images/20160818/tooopen_sy_175833047715.jpg" width="355" height="150"/>
            </swiper-item>
        </swiper>
        <button bindtap="switchAutoplay">
            切换 autoplay 状态
        </button>
    </view>
```

之后我们在 swiper.js 中编写如下代码：

```
Page({
    data:{
        isAutoplay:false
    },
    switchAutoplay:function(){
        this.setData({
            isAutoplay:!this.data.isAutoplay
        })
    },
    picChange:function(){
        console.log('焦点图已切换')
    }
}
)
```

在该代码中，我们在 data()方法里设置了绑定变量 isAutoplay 的初始值为 false，并在 switchAutoplay 方法中调用 this.setData()方法，改变绑定的 isAutoplay 的值，!this.data.isAutoplay 会每次与结果的值取反。

然后点击“调试”按钮进入调试模式，运行效果如图 4.11 所示。

图 4.11

点击“切换 autoplay”状态按钮，焦点图会自动切换，并在每次切换时在控制台打印出“焦点图已切换”的字样，再次点击“切换 autoplay”状态按钮，会停止自动切换焦点图。

至此，我们实现了一个简易但功能完备的“焦点图”效果，读者可以在此例基础上多修改一些属性，来体验不同属性之间的差异，继续熟悉该组件的用法。

4.5　基础内容

本章介绍三个比较基础的组件，分别是图标组件 icon，文本组件 text，以及进度条组件。这三个组件相对比较简单，但都比较常用，相信读者能通过本章内容熟练掌握其用法。

4.5.1　图标组件 icon

小程序提供的 icon 用来显示一些微信风格的默认图标，使编写的程序风格与微信保持一致。

icon 组件的使用语法为：

```
<icon type="success"/>
```

官方文档提供了其属性列表，如表 4.11 所示。

表 4.11　icon 组件的属性

属性名	类　型	默认值	说　明
type	String		icon 的类型，有效值：success, success_no_circle, info, warn, waiting, cancel, download, search, clear
size	Number	23	icon 的大小，单位 px
color	Color		icon 的颜色，同 css 的 color

可以根据自己的情况来决定是否创建一个新的目录，为方便起见，直接在上一个项目的 swiper.wxml 中增加一个 icon 组件：

```
<icon type="success"/>
```

保存后，即可在工作区域看见图标的效果，如图 4.12 所示。

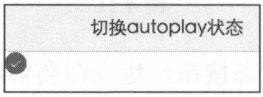


图 4.12

可以看到其除了属性列表中介绍的默认字体大小 23 外，还有一个默认的 color 为绿色。

也可以加入其他属性来体会一下图标组件的用法，如：

```
<icon type="success" size="42" color="#B0C4DE" />
```

保存后可以看到图标的效果变为如图 4.13 所示。

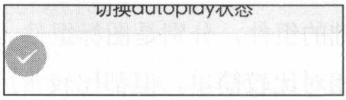


图 4.13

为了体会所有图标组件的所有类型，我们结合 WXML 的列表渲染功能，来编写一段代码展现一下所有的图标类型，顺便巩固一下 WXML 列表渲染部分的内容。

为了代码的清晰，我们在 pages 目录下新建一个 basic 目录，并在 basic 目录下分别创建 basic.wxml 文件和 basic.js 文件，然后在 app.json 的 pages 数组中新增

"pages/basic/basic"，而且需要将其放在首个，以保证代码先以 basic 为入口运行，并在 basic.wxml 中编写如下代码：

```
<view>
    <icon wx:for="{{type}}" type="{{item}}" size="45"/>
</view>
```

在该代码中，我们用 icon 标签创建了一个图标组件，并将其属性 type 绑定为数据源默认的循环元素{{item}}，然后使用列表渲染属性 wx:for，绑定循环数组 type。

在 basic.js 文件中编写如下代码：

```
Page({
    data:{
        type: [
        'success',  'info',  'warn',  'waiting',  'safe_success',
'safe_warn',
        'success_circle',  'success_no_circle',  'waiting_circle',
'circle', 'download',
        'info_circle', 'cancel', 'search', 'clear'
        ]
    }
})
```

该代码唯一做的就是指定了 type 数组的值，即我们通过数组包含了表 4.11 中所述的所有图标类型，保存后，即可在视图区看到效果，如图 4.14 所示。

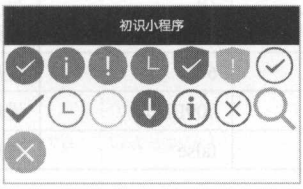


图 4.14

对于图标组件主要就是 type 属性的使用，相对还是比较简单的。

4.5.2　文本组件 text

文本组件就是用来显示一段文字，其标签语法为：

```
<text>hello</text>
```

*这个组件也没有自己的专有属性，其特点是可以支持转移字符“\”，并且需*要注意组件内只支持<text/>标签的嵌套，不能有其他组件标签的嵌套，如果有其他非 text 的组件标签，虽然不会报错，但是不会有任何呈现，而是会被忽略。

我们可以在 basic.wxml 文件中加入如下代码：

```
<text>这里是第一行，\n 这里却是第二行</text>
```

运行后则会看到转义字符\n 完成了换行，如图 4.15 所示。

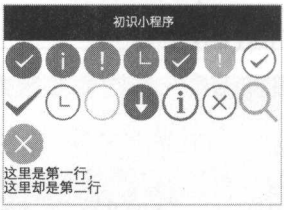


图 4.15

4.5.3 进度条组件 progress

相信任何人对进度条都不会陌生，其适用场景也不需过多的介绍，我们先贴出微信官方文档提供的属性列表（见表 4.12 所示），以方便读者查阅。

表 4.12 progress 组件的主要属性

属性名	类 型	默认值	说 明
percent	Float	无	百分比 0~100
show-info	Boolean	false	在进度条右侧显示百分比
stroke-width	Number	6	进度条线的宽度，单位 px
color	Color	#09BB07	进度条颜色
active	Boolean	false	进度条从左往右的动画

进度条组件 progress 的标签语法为：

```
<progress percent="20" />
```

参照表 4.12 所示的属性列表就很容易理解，下面通过一个完整案例来加深理解。

为了代码的清晰，将 basic.wxml 文件里的代码改写为：

```
<view>
    <progress percent="{{percent}}"
        style="margin:300rpx 40rpx 100rpx 40rpx" />

    <button bindtap="startPercent">重新加载进度条</button>
</view>
```

这里通过<progress />和<button/>标签创建了一个进度条和一个按钮，进度条组件的 percent 属性通过绑定的数据源{{percent}}传入，按钮组件绑定了 startPercent 事件，紧接着将原来的 basic.js 文件代码改写为：

```
Page({
    data:{
        percent:0
    },

    /**
     * 开始进度条方法
     */
    startPercent:function(){
        var that=this
        that.setData({
            percent:0
        })
        var interval=setInterval(function(){
            if(that.data.percent<100){
                that.setData({
                    percent:that.data.percent+1
                })
            }else{
                clearInterval(interval)
            }
        },30)
    },

    /**
     * 生命周期函数，用来在页面渲染完成后调用 startPercent 方法
     */
    onReady:function(){
```

```
            this.startPercent()
        }
    })
```

*该段代码首先在 data 对象中绑定了 precent 变量，并赋予了初始值 0，然后*定义了 startPercent 方法，该方法绑定到了 wxml 文件中的 button 中，实现了点击按钮就重新开始一次进度条进度，代码通过 setInterval 间隔函数循环触发 setData() 方法来不断累加绑定变量 percent 的值，if 的判断是用在 percent 值达到 100 时删除间隔函数。之后为了程序在一开始就运行，就在生命周期 onReady()中调用了一次 startPercent 方法。

保存并运行上述代码，小程序运行的效果如图 4.16 所示。

在调试环境下重新编译后，进度条会从左侧填充到右侧，如果点击按钮“重新加载进度条”，会再次从左侧开始填充。

读者一定注意到了，progress 组件的属性当中有一个 active，设置该属性值为 true 以后，只需设置一次 percent 的值，小程序就会以动画形式自动从头填充到这个位置，但是这在多数实际开发中并不适用，正式开发中通常都是按任务的进度来动态展示进度条的位置，以告知用户当前任务的处理进度，而启用 active 属性后，每次设置 percent 的值，都会从头填充，这并不符合常见的应用场景，除非是展示一个比例。

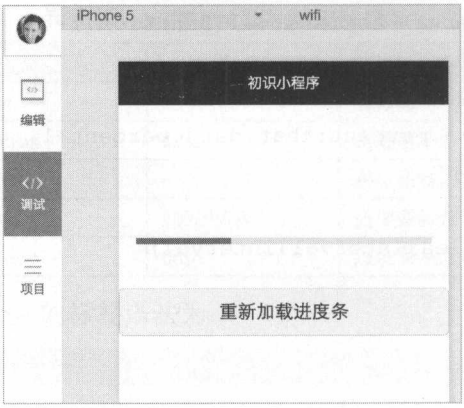


图 4.16

其他属性的方法比较简单，读者感兴趣可以自行尝试修改和体会。

4.6　导航

到目前为止，我们在项目中已经建立了三个目录来介绍不同的例子。而每次启动一个例子，都是通过修改配置文件 app.json 改变启动页面来实现的，读者也可以制作一个导航页，通过链接来导航到不同的页面，从而实现这个功能，这就需要用到本节要介绍的 navigator 页面链接组件。

组件的标签语法非常简单：

```
<navigator url="../../pages/index/index" hover-class="navigator-hover">跳转到新页面</navigator>
```

官方提供了如表 4.13 所示的属性列表。

表 4.13　navigator 组件的主要属性

属性名	类　型	默认值	说　明
url	String		应用内的跳转链接
redirect	Boolean	false	是否关闭当前页面
hover-class	String	navigator-hover	指定点击时的样式类，当 hover-class="none"时，没有点击态效果

总共三个属性，其中比较重要的 url 和 redirect 属性依然通过一个实例来说明。

既然说到了导航页，这里就来做一个导航页。

（1）在 pages 目录下新建一个名为 navigate 的目录，并依次在该目录创建 navigate.wxml 和 navigate.js 文件，然后在 app.json 文件的 pages 项数组中增加如下项，并保证是数组第一个元素：

```
"pages/navigate/navigate",
app.json 文件完整的代码为：
{
  "pages":[
    "pages/navigate/navigate",
    "pages/basic/basic",
    "pages/swiper/swiper",
    "pages/index/index"
  ],
  "window": {
     "navigationBarTitleText": "初识小程序"
```

```
    }
}
```

（2）如果此时保存运行，小程序视图会显示一个空白页面，相信读者也明白原因：小程序的入口页面改为了空的 navigate。

我们在 navigate.wxml 文件中编写如下代码：

```
<view>
    <navigator url="../../pages/index/index"
    hover-class="navigator-hover">首页</navigator>

    <navigator url="../../pages/swiper/swiper"
    hover-class="navigator-hover">焦点图</navigator>

    <navigator url="../../pages/basic/basic"
    redirect
    hover-class="navigator-hover">基本视图</navigator>
</view>
```

（3）保存代码，导航页面就完成了，效果如图 4.17 所示。

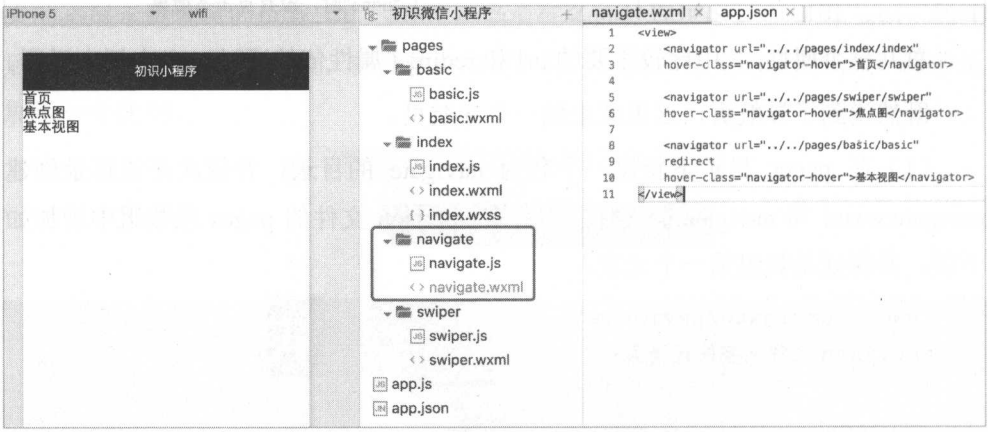


图 4.17

这里需要特别注意的是 url 这个属性的值，在定位其他页面时的路径是根据当前目录的路径，因此需要用两个../../把位置定位到根目录，再通过 pages/basic/basic 指定到目标文件，并且指定的文件必须是在 app.json 文件中的 pages 数组中配置过的，否则无法导航过去。

现在依次点击这两个链接，都会按预期的导航到相应页面，而前两个链接则分别导航到了我们之前的实例首页和焦点图页，如图 4.18 所示。

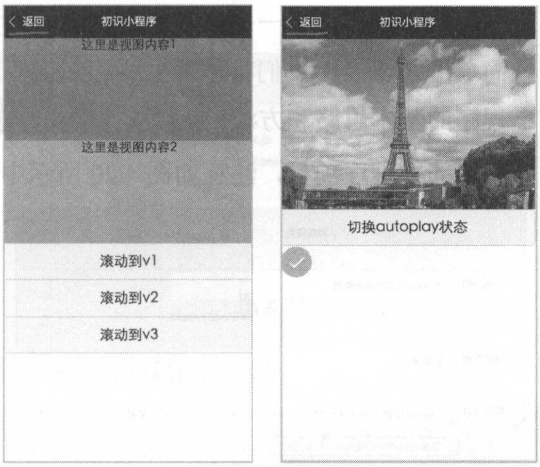


图 4.18

在这两个页面中，页面顶部导航会出现一个返回的按钮，点击可以回到首页导航页面。而如果点击“基本视图”链接，则会导航到如图 4.19 所示。

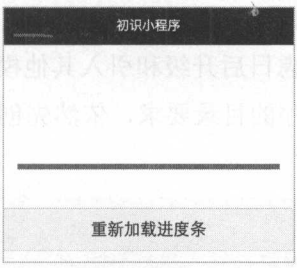


图 4.19

可以看到跳转到该页面后，左上角没有了返回按钮，是因为加入了属性 redirect，这种 boolean 类型的属性可以直接使用其属性名，redirect 就等价于 redirect="true "。由此可见，当设置了 redirect 属性为 true ，跳转链接后会关闭当前页面，即跳转过去后用户无法返回当前页面。

这两种跳转类型读者可以根据自己的需要来使用。

4.7 手把手教你做 Demo——简易通讯录

本章手把手环节，我们来一起开发一个简易的通讯录，且只有增、删功能，但通过开发这样一款简单的通讯录，我们将本章主要的知识梳理一遍，读者也可以进一步熟悉开发小程序的基本思路和方法。

（1）首先创建一个新的小程序项目，注意如图 4.20 所示中方框标注的区域。

图 4.20

（2）创建后，由于项目目录为空，会出现报错，这时不用理会，虽然是一个简单的项目，我们也需要考虑日后升级和引入其他模块的情况，因此也需要合理的规划目录结构。按照小程序的目录要求，依然先创建一个 app.json 文件，并编写如下代码：

```
{
  "pages":[
    "view/index/index"
  ],
  "window": {
     "navigationBarTitleText": "简易通讯录"
    }
}
```

（3）然后按照 pages 数组的项，创建一个 view 目录，并在 view 目录下，创建一个 index 目录，并在 index 目录下分别创建 index.js 与 index.wxml 文件。

目录结构与 app.json 文件的内容如图 4.21 所示，至此，小程序可以不报错的运行起来，读者可以回顾一下前面章节介绍的小程序最小结构。

另外从本节也已经了解到，微信开发者工具在最新版本中，会自动创建所需的目录结构及文件，读者可能会自动得到与图 4.21 不完全一样的文件结构，但是不影响本实例的开发。

图 4.21

（4）在 index.js 文件中编写如下代码:

```
Page({
    data:{
        contactData:[
            {
                id:'1',
                name:'汪鑫',
                phone:'18612345678',
                birthday:'1990'
            },
            {
                id:'2',
                name:'李梅',
                phone:'18612345679',
                birthday:'1990'
            },
            {
                id:'3',
                name:'马云',
                phone:'18612345681',
                birthday:'1971'
            },
            {
                id:'4',
                name:'冰冰',
                phone:'18612345681',
```

```
            birthday:'1984'
        },
        {
            id:'5',
            name:'大史',
            phone:'18612345681',
            birthday:'1977'
        },
        {
            id:'6',
            name:'卫宁',
            phone:'18612345681',
            birthday:'1988'
        },
        {
            id:'7',
            name:'小李',
            phone:'18612345681',
            birthday:'1987'
        },
    ]
}
})
```

这段代码只是在 data 中定义了一个数组 contactData，其结构由 id、name、phone、birthday 四个字段组成，数组的数量读者也可以任意修改，为了简化计算，birthday 项暂时只用了年份。

（5）紧接着，在 index.wxml 文件中编写如下代码：

```
<view>
    <view wx:for="{{contactData}}" wx:key="id">
        <text>{{item.id}}</text>
        <text>| {{item.name}}</text>
        <text>| {{item.phone}}</text>
        <text>| {{item.birthday}} </text>
        | <icon data-index="{{index}}" type="cancel" size='14' />
    </view>
</view>
```

这段代码中，使用 wx:for 属性遍历了我们在 index.js 文件中定义的 contactData

数组，并用 wx:key 属性关联了 contactData 数组项中唯一属性 id，这样该组件在遍历时会与数组的顺序对应起来，然后我们使用 icon 组件实现了删除按钮，在这里使用 data-index="{{index}}"的用意是为了在遍历时把 contactData 数组的索引值放在该属性中，之后我们绑定事件后就可以方便地获取该属性的值。

（6）保存后视图区域可以看到如图 4.22 所示的效果。

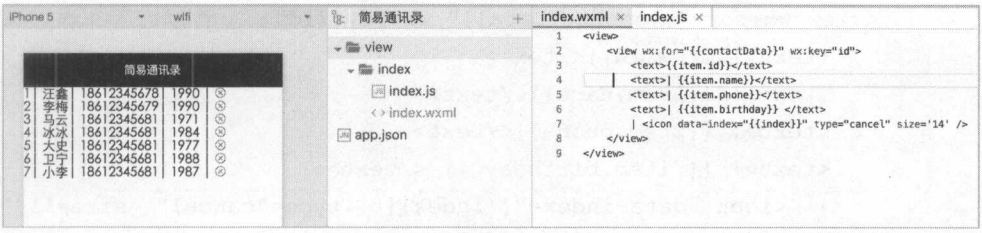


图 4.22

可以看到通过 wx:for 已经显示出了在 index.js 文件中定义的 contactData 数组的内容。

现在进入调试模式，然后在右侧控制台点击 Wxml 标签，即可在控制台看到小程序最终展示的页面标签结构，这儿依次展开 page-view-view 标签，并再展开最后一级 view 标签，可以看到如图 4.23 所示的效果。

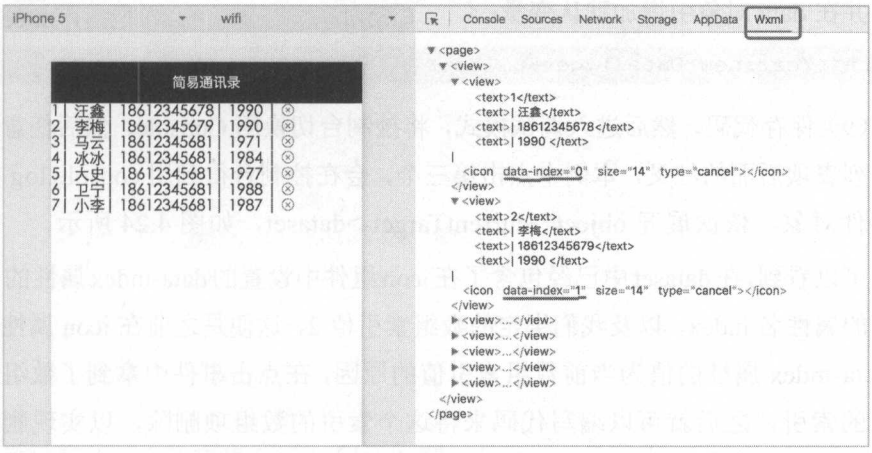


图 4.23

在 Wxml 中，我们已经可以直观地看到通过 wx:for 属性实现的列表渲染 wxml 结构，即遍历了 contactData 数组后的效果，其中比较重要的是图 4.23 所示中用粗线标出的 data-index 属性的效果，可以看出是依次绑定了数组的索引，笔者可以

多展开几个 view 标签来观察，可以加深对 wx:for 实现列表渲染的理解。

目前只是展示了在数组 contactData 中的数据内容，下面继续实现删除功能。

（7）修改 index.wxml 文件中的代码，在 icon 组件中中加入事件绑定：bindtap="deleteMe"，完整 index.wxml 文件的代码为：

```
<view>
    <view wx:for="{{contactData}}" wx:key="id">
        <text>{{item.id}}</text>
        <text>| {{item.name}}</text>
        <text>| {{item.phone}}</text>
        <text>| {{ item.birthday }} </text>
        | <icon data-index="{{index}}" type="cancel" size='14'
bindtap="deleteMe" />
    </view>
</view>
```

（8）在 index.js 中增加绑定事件 deleteMe 的实现，具体代码为：

```
deleteMe:function(e){
    console.log(e)
}
```

并在 data 对象中增加默认变量：

```
thisYear:new Date().getFullYear(),
```

（9）保存代码，然后进入调试模式，将控制台切换回 Console，然后任意点击一个列表项后面的红叉，本例中点击第三个，会在控制台得到用 console.log 打印的事件对象，依次展开 object->currentTarget->dataset，如图 4.24 所示。

可以看到，在 dataset 中已经包含了在 icon 组件中设置的 data-index 属性的 data-之后的属性名 index，以及我们绑定的数组索引值 2，这便是之前在 icon 属性中设置 data-index 属性的值为当前数组索引值的原因，在点击事件中拿到了数组该项对应的索引，之后就可以编写代码来将这个索引的数组项删除，以实现删除的功能。

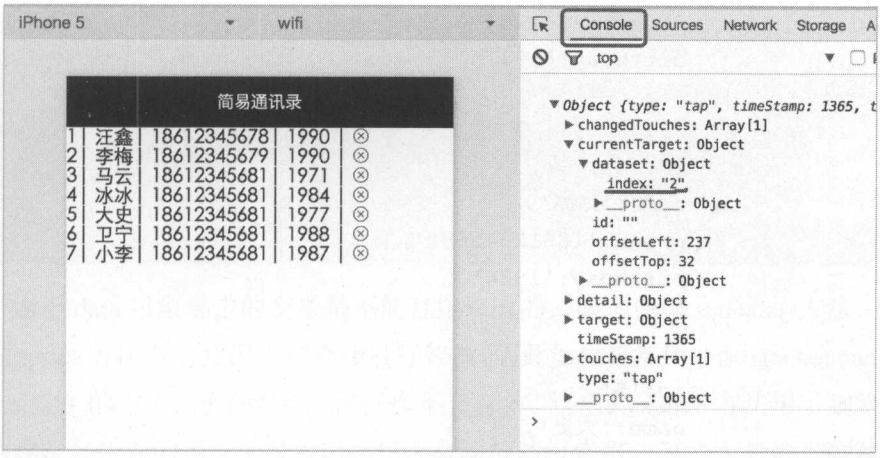


图 4.24

该界面可能会令读者不忍直视，不过不用在意，我们会在后续章节讲到其他组件后，继续优化该实例。重要的是通过循序渐进的讲解，能让读者更好地掌握小程序的开发。

（10）下面通过修改 deleteMe 方法来实现删除功能，修改后 index.js 文件的完整代码为：

```
Page({
data:{
   thisYear:new Date().getFullYear(),
      contactData:[
         {
            id:'1',
            name:'汪鑫',
            phone:'18612345678',
            birthday:'1990'
         },
         {
            id:'2',
            name:'李梅',
            phone:'18612345679',
            birthday:'1990'
         },
         {
            id:'3',
            name:'马云',
```

```
            phone:'18612345681',
            birthday:'1971'
        },
        {
            id:'4',
            name:'冰冰',
            phone:'18612345681',
            birthday:'1984'
        },
        {
            id:'5',
            name:'大史',
            phone:'18612345681',
            birthday:'1977'
        },
        {
            id:'6',
            name:'卫宁',
            phone:'18612345681',
            birthday:'1988'
        },
        {
            id:'7',
            name:'小李',
            phone:'18612345681',
            birthday:'1987'
        },
    ]
},
deleteMe:function(e){
    console.log(e)
    //由于不能直接操作初始化对象 data 中的值,我们需要创建一个副本进行修改
    var newContactData=this.data.contactData
    //通过事件对象获得当前被点击项的数组索引
    var index=e.currentTarget.dataset.index
    //调用数组的 splice 方法删除相应索引的项
    newContactData.splice(
        index ,1
    )
```

```
            //重新将修改后的数组，通过 setData 方法赋值到初始化对象 data 的
contactData 数组中
            this.setData({
                contactData:newContactData
            })
        }
    })
```

由于 data 对象绑定的变量是不能直接修改的，必须使用 setData()方法，而数组的 splice 方法会对数组本身直接进行修改，因此如果直接用 this.data.contactData.splice(1,1) 的方式来删除数组，该操作不会有效。这里的 splice 方法用于删除数组的内容，其传入的第一个参数用来指定删除的数组索引，第二个参数为删除的长度。该方法功能十分强大，若读者不是很熟悉可以参阅相关资料。本例中先通过一个变量 newContactData 来复制一份 contactData 的值，然后再通过 newContactData.splice(index,1)来删除当前索引的数组项，最后再通过 setData()方法，将修改后的 newContactData 值赋给 contactData，最终实现对绑定的数据修改。

需要留意的是，我们在这段代码中的 data 对象部分又加入了 thisYear:new Date().getFullYear()，是为了得到当前时间的年份，这样就可以很方便地在 wxml 中通过简单的减法算出联系人的年龄，因此还需要将 index.wxml 文件中的绑定{{ item.birthday}}修改为{{thisYear-item.birthday}}，这里便使用了 data 绑定的 this year 减去当前数组项的 birthday 字段，计算出联系人的年龄。

数据绑定开发的重要特点，就是只需考虑修改绑定后的数据，不用关心视图的表现逻辑，这在很大程度上能够提高开发效率，但同时需要一些思维的转变。

（11）保存代码，并进入调试模式，然后尝试点击第 4 条后面的删除按钮，之后的效果如图 4.25 所示。

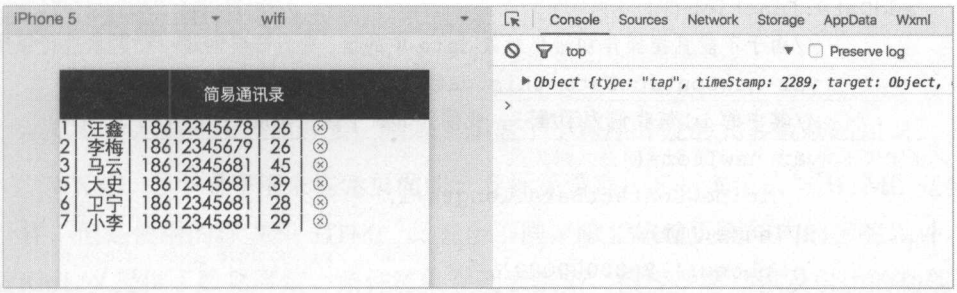


图 4.25

可见第 4 条记录已经被我们删除。

接下来，我们继续实现记录的新增功能。由于我们还未学习小程序表单的使用，这里先实现固定数据的新增。

（12）在 index.wxml 文件中新增一个 button 标签，并用 bindtap 事件绑定 addData 事件，读者也许也已经发现了，bindtap 属性是十分常用的一个属性，希望读者也能够熟练地掌握其用法。然后将列表渲染的 view 组件放入一个 scroll-view 标签中，以便新增了多条数据后能够在这个视图中滚动，加入后完整的代码如下：

```
    <view>
        <scroll-view scroll-y style="height:300rpx">
            <view wx:for="{{contactData}}" wx:key="id">
                    <text>{{item.id}}</text>
                    <text>| {{item.name}}</text>
                    <text>| {{item.phone}}</text>
                    <text>| {{item.birthday}} </text>
                    |    <icon    data-index="{{index}}"    type="cancel"
size='14' bindtap="deleteMe" />
            </view>
        </scroll-view>

        <button style="margin-top:50rpx;height:70rpx;font-size:29rpx"
bindtap="addData">
            新增
        </button>
    </view>
```

其中 style 属性是一些简单的样式，读者也可以根据自己的偏好进行修改。

（13）在 index.js 中，增加已绑定的 addData 方法的实现，具体代码如下：

```
    addData:function(){
            //由于不能直接操作初始化对象 data 中的值，我们需要创建一个副本进行修改
            var newContactData=this.data.contactData
            //其中的 id 值我们为了唯一，我们为其赋予数组长度+1
            var newItem={
                id:newContactData.length+1,
                name:'新人',
                phone:'18600000000',
                birthday:'1970'
```

```
        }
        newContactData.push(newItem)
        this.setData({
            contactData:newContactData
        })
    }
```

由于其添加方法与 deleteMe 一致，这里就不贴出完整代码了。

其中因为用到了数组的 push 方法，也是会对数组直接进行修改，因此也是采用了与 deleteMe 方法一样的思路，将 contactData 赋值给新的局部变量 newContactData，在 newContactData 中通过 push 方法增加新的项后，再通过 setData 方法将绑定的 contactData 变量赋值为新的局部变量 newContactData 的值。

另外因为这里我们用不到事件对象，因此没有传入事件对象参数。

（14）保存代码后，进入调试模式，点击几次“新增”按钮，就实现了对通讯录数据的新增，并且新增了多条后会出现滚动条，效果如图 4.26 所示。

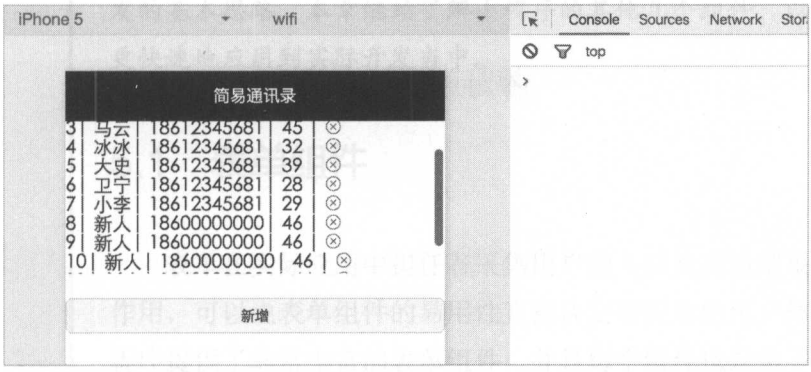


图 4.26

4.8　本章要点总结

本章学习的知识点比较多，基本上涵盖了小程序开发的大多数基础知识。

WXML 是小程序用来构建页面的一套标签语言，其基本语法与 HTML 完全一样，但是使用的标签和 HTML 又完全不同，除了构建页面结构的标签以外，WXML 又提供了数据绑定、条件渲染、列表渲染、模板、事件以及引用的功能，对于数据绑定的开发模式，这些特性是强有力的支持。

WXSS 即是小程序的 CSS 样式，用于描述 WXML 的组件样式，其用法与 CSS 完全一样，只是 WXML 为我们提供了新的尺寸单位 *rpx*，可以方便地实现屏幕的自适应。并且支持样式的导入，方便我们把常见的样式抽象出来做成一个公用样式模块。

之后又熟悉了几个比较关键的基础组件用法，通过实例进一步加深了我们对这几个组件、事件绑定以及列表渲染的用法，体会了小程序开发的关键思路，对我们后续章节继续深入学习将会有很大帮助。

第5章

小程序开发实战：全面掌握小程序组件

通过前一章对小程序基础的了解，我们已经熟悉了小程序开发的基本思路，本章继续了解小程序的其他几个组件，以便能够更快速地应用到实际开发当中。

5.1 表单组件

表单在实际应用中担任着采集用户输入以及反馈结果的关键作用，可以说表单组件的易用性直接决定着应用的用户体验。小程序提供了非常丰富的表单组件，并且每个组件也封装了非常实用的功能，很适合国内应用的开发需求。

5.1.1 按钮组件button

到目前为止，button 组件对我们而言应该说已经不陌生了，之前介绍的几个例子中，我们也已经有多次接触，其标签语法也非常简单：

```
<button> </button>
```

但是目前为止我们只是使用了 button 最基础的功能，实际上 button 组件提供了非常丰富的属性，官方文档提供的属性如表 5.1 所示。

表 5.1　属性列表

属性名	类　型	默认值	说　明
size	String	default	有效值 default, mini
type	String	default	按钮的样式类型，有效值 primary, default, warn
plain	Boolean	false	按钮是否镂空，背景色透明
disabled	Boolean	false	是否禁用
loading	Boolean	false	名称前是否带 loading 图标
form-type	String	无	有效值：submit, reset，用于 <form/> 组件，点击分别会触发 submit/reset 事件
hover-class	String	button-hover	指定按钮按下去的样式类。当 hover-class="none" 时，没有点击态效果

这些属性都非常容易理解，其中只有 form-type 是需要 from 组件来配合使用的。即如果是指定为 submit 类型，则点击该按钮时，会触发 form 组件的 bindsubmit 事件，而如果指定为 reset，则点击该按钮会触发 form 组件的 bindreset 事件。至于详细用法，会在稍后介绍 form 组件时详细阐述。

看几个例子加深一下对其他属性的理解。

在任意 wxml 文件中编写如下代码：

```
<button size="mini">小按钮</button>
<button >默认按钮</button>
```

保存后可以看到如图 5.1 的效果。

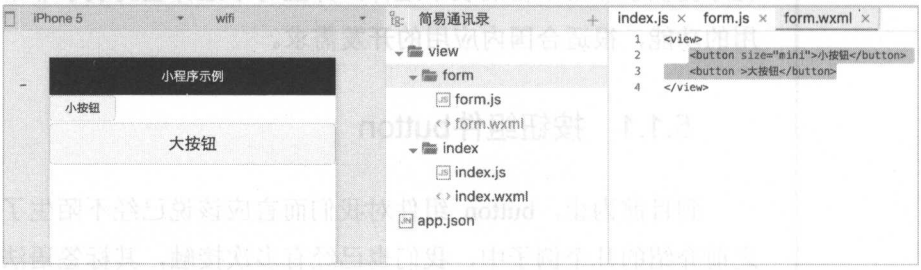


图 5.1

size 属性可以方便地实现一个比较小的按钮，且省略一部分 wxss 样式代码。

另外一个比较重要的属性是 loading 属性，加入该属性的按钮效果如图 5.2 所示。

图 5.2

其主要的用途实际上是在执行一个可能比较耗时的操作时，给用户一个等待的提示。例如需要访问一个网络请求，就可以先用属性 disabled 来禁用按钮，然后再用一个 loading 属性来提示用户等待，以避免用户多次点击按钮产生多次请求。下面通过一个实例来模拟这个场景。

在 wxml 文件中编写如下代码：

```
<view>
<button loading="{{loading}}" disabled="{{disabled}}" bindtap="switchState">提交
</button>
</view>
```

然后在 js 中编写如下代码：

```
Page({
    data:{
        loading:false,
        disabled:false
    },
    switchState:function(){
        //使用日志输出来了解按钮的有效点击次数
        console.log('点击了一次')
        //通常需要在请求异步操作时保存一下当前的 this 环境，以在异步方法内访问到当前的 this 环境
        var that=this
        //这里通过修改绑定的 loading 和 disabled 初始值来将按钮的状态切换为禁用，并显示 loading 图标
        this.setData({
            loading:true,
            disabled:true
        })
        //我们通过创建一个 setTimeout 来模拟一个网络请求
        var timer=setTimeout(function(){
            //通常这个状态的切换会写在类似网络请求的回调中，以切换回初始的状态
```

```
            that.setData({
                loading:false,
                disabled:false
            })

        },1000)
    }
})
```

这段代码采用 setTimeout 方法模拟了一个网络请求的场景。先用组件的 loading="{{loading}}" 和 disabled="{{disabled}}"设置绑定了 loading 和 disabled 初始值为 false，并在绑定的点击事件 switchState 中切换按钮的状态为 loading 为 true，disable 也为 true，同时用 setTimeout 方法创建一个 1 秒后调用的函数，并在函数内用 setData 重新将绑定的 lading 和 disabled 初始值改为 false，完成状态的恢复。

笔者在保存后切换到调试模式，可以通过点击按钮的方式来感受这两个属性相结合带来体验上的优化，如图 5.3 所示。

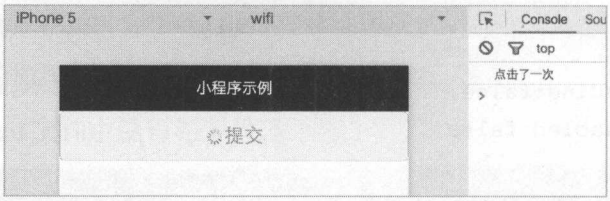


图 5.3

用户在点击后会很直观地感受到程序正在执行点击之后的操作，并且如果此时再次点击则不会有任何响应。实现这样的功能使用 button 组件自身的这两个属性，结合数据绑定的方式，实现起来非常方便和清晰。

5.1.2 标签组件 label

该标签的语法为：

```
<label for=""></label>
```

而属性也只有一个 for，用来指定一个组件的 id 进行绑定。绑定后在该组件上操作的事件会触发绑定的组件事件。

在 wxml 文件中编写如下代码：

```
<label>
  label 组件示例
  <button id="v_clickme" bindtap="clickme">点一下我试试？ </button>
</label>
```

然后在 js 文件的 Page()函数中增加如下方法：

```
    clickme:function(){
        console.log('我被点了')
}
```

保存后我们进入调试模式，点击“点一下我试试？”按钮，控制台会打印出“我被点了”的日志。label 的特色是，当我们点击写在 label 里的文字内容“label 组件示例”时，也一样触发了通过 id 绑定的 button 的 clickme 事件，如图 5.4 所示。

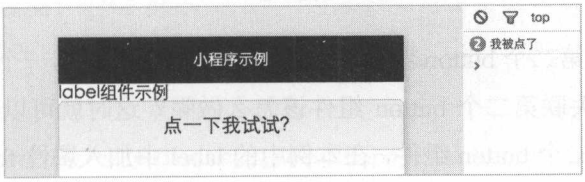


图 5.4

这一点才是 label 组件的主要功能所在，可见对其包含的组件，会自动绑定第一个组件，并触发第一个组件的事件，如果需要触发其他组件的事件，则需要用 for 属性来指定需要绑定的组件 id，但是这里需要注意的是，label 在写本书时，只支持绑定<button/>, <checkbox/>, <radio/>, <switch/>这四个组件，绑定其他组件是无效的。

我们在本例中增加一个 button 组件，将 wxml 文件的代码修改为：

```
<view>
    <label>
label 组件示例
<button id="v_clickme" bindtap="clickme">点一下我试试？ </button>
      <button id="v_clickme2" bindtap="clickme2">我是第二个按钮
</button>
    </label>
</view>
```

然后在 js 文件中的 Page()函数中新增一个方法：

```
    clickme2:function(){
        console.log('第二个按钮被点')
    }
```

保存并运行，然后点击文字标题“label 组件示例”，控制台打印出日志如图 5.5 所示。

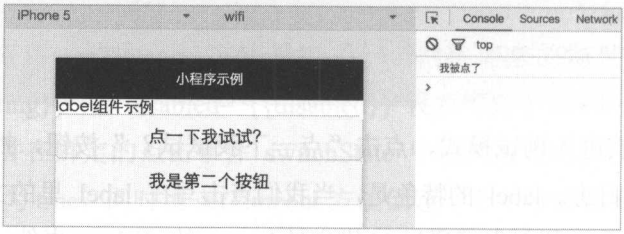


图 5.5

可见增加了第二个 button 组件后，label 依然默认关联第一个 button 组件，如果需要让 label 关联第二个 button 组件该怎么做呢？这时就可以借助 label 的 for 属性，来指定第二个 button 组件，在本例中的 label 中加入属性 for="v_clickme2"，wxml 文件的完整代码则为：

```
<view>
    <label for="v_clickme2">
        label 组件示例
        <button id="v_clickme" bindtap="clickme">点一下我试试？
</button>
        <button id="v_clickme2" bindtap="clickme2">我是第二个按钮
</button>
    </label>
</view>
```

保存代码，进入调试模式，为了效果更直观，点击左侧工具栏的“编译”按钮，重新编译一次代码，然后点击“label 组件示例”标题字样，运行的效果如图 5.6 所示。

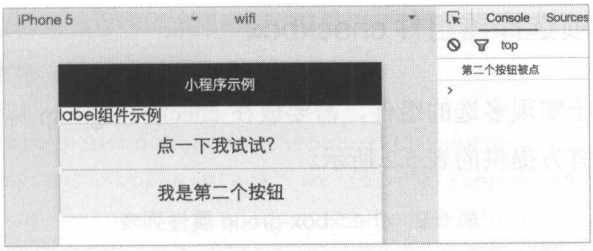


图 5.6

可见通过绑定后，点击 label 组件触发了绑定的组件 bindtap 事件。

我们接着修改 wxml 文件，将第二个 button 移到 label 标签外，修改后的 wxml 文件的代码为：

```
    <view>
        <label for="v_clickme2">
            label 组件示例
            <button id="v_clickme" bindtap="clickme">点一下我试试?
</button>
        </label>
        <button id="v_clickme2" bindtap="clickme2">我是第二个按钮
</button>
    </view>
```

保存代码并进入调试模式，再点击右侧工具栏的“编译”按钮，然后再次点击“label 组件示例”标题字样，运行的效果如图 5.7 所示。

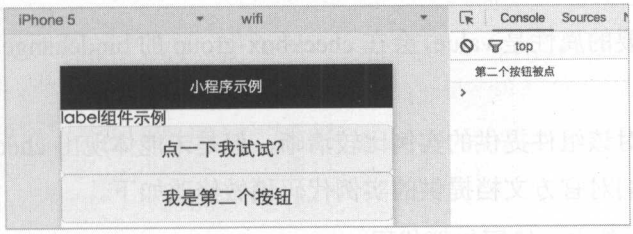


图 5.7

很明显，通过 label 标签的 for 属性指定组件的 id 后，即使组件不在 label 标签内，也会触发绑定的组件的事件。

这里还需要注意一下，label 组件为内联组件。

5.1.3 多项选择器组件 checkbox

多选框是用于实现多选的组件，需要放在 checkbox-group 标签内使用，其属性只有一项，如官方提供的表 5.2 所示。

表 5.2 checkbox-group 属性列表

属性名	类 型	默认值	说 明
bindchange	EventHandle		<checkbox-group/>中选中项发生改变是触发 change 事件，detail = {value:[选中的 checkbox 的 value 的数组]}

这里的 bindchange 事件为 checkbox-group 的特殊事件，会在事件对象中传入特殊的 detail 属性。

其标签的语法为：

```
<checkbox-group > </checkbox-group>
```

checkbox-group 的主要作用是对多选框组件进行分组，这种处理机制更为直观。

而 checkbox 组件的主要属性，官方说明如表 5.3 所示。

表 5.3 checkbox 属性列表

属性名	类 型	默认值	说 明
value	String		<checkbox/>标识，选中时触发<checkbox-group/>的 change 事件，并携带 <checkbox/> 的 value
disabled	Boolean	false	是否禁用
checked	Boolean	false	当前是否选中，可用来设置默认选中

其中最重要的属性是 value，会在 checkbox-group 的 bindchange 事件触发时以事件对象传入。

官方文档对该组件提供的实例比较清晰，但是未能体现出 checkbox-group 的分组作用，我们对官方文档提供的实例代码稍做修改如下。

在 wxml 文件中，编写如下代码：

```
<checkbox-group bindchange="checkbox1Change">
  <label style="display:block" wx:for="{{items1}}">
    <checkbox value="{{item.name}}" checked="{{item.checked}}"/>
{{item.value}}
  </label>
</checkbox-group>
```

```
    <view  style="margin-top:30rpx;margin-bottom:30rpx;border-top:1px
solid #000"></view>

    <checkbox-group bindchange="checkbox2Change">
      <label style="display:block" wx:for="{{items2}}">
        <checkbox  value="{{item.name}}"  checked="{{item.checked}}"/>
{{item.value}}
      </label>
    </checkbox-group>
```

在该代码中，我们在官方实例的基础上实现了两组多选组件，第一个为国家多选框，第二个为地区多选框。

每个 checkbox-group 组件在 bindchange 属性中分别绑定了 checkbox1Change 和 checkbox2Change 事件。

这里用到了上一节介绍的 label 组件，用于将 checkbox-group 组件触发的事件继续应用在其绑定的{{item.value}}内容中，如果这里改为 view，则多选框的多选项文字部分则无法触发 bindchange 事件，用户体验就会大打折扣。在 checkbox-group 这种多选框应用中属于是通用的做法，希望读者多多练习以加强记忆。

接着在 js 文件中编写如下代码：

```
Page({
  data: {
    items1: [
      {name: 'USA', value: '美国'},
      {name: 'CHN', value: '中国', checked: 'true'},
      {name: 'BRA', value: '巴西'},
      {name: 'JPN', value: '日本'},
      {name: 'ENG', value: '英国'},
      {name: 'TUR', value: '法国'},
    ],
    items2: [
      {name: 'BEIJING', value: '北京', checked: 'true'},
      {name: 'SHANGHAI', value: '上海'},
      {name: 'SHANDONG', value: '山东'},
      {name: 'GUANGDONG', value: '广东'},
      {name: 'SHENZHEN', value: '深圳'},
      {name: 'HANGZHOU', value: '杭州'},
```

```
        ],
        selectedItems1:[],
        selectedItems2:[]
    },
    checkbox1Change: function(e) {
      //通过事件对象的 e.detail.value 获得当前选中的项的 value 属性值的组合
      var selectedItems=e.detail.value
      console.log('checkbox1 发生 change 事件,携带 value 值为:', selectedItems)
      //通过 setData 将当前选中项的值保存在默认初始化状态 selectedItems1 对象中
      this.setData({
         selectedItems1:selectedItems
      })
      console.log('data 中 selectedItems1 的值:',this.data.selectedItems1)
    },
    checkbox2Change: function(e) {
      //通过事件对象的 e.detail.value 获得当前选中的项的 value 属性值的组合
      var selectedItems=e.detail.value
      console.log('checkbox2 发生 change 事件,携带 value 值为:', e.detail.
value)
      //通过 setData 将当前选中项的值保存在默认初始化状态 selectedItems1 对象中
      this.setData({
         selectedItems2:selectedItems
      })
      console.log('data 中 selectedItems2 的值:',this.data.selectedItems2)
    }
  })
```

这里定义了两个对象数组，分别为 items1 与 items2，其数组项的结构完全一致，包含字段 name 与 value 项，并在需要的项中设置了默认需要选中的 checked 属性，通过与其对应的属性绑定 checked="{{item.checked}}"，可见小程序 wxml 中绑定不存在的 data 对象是不会报错的。

保存代码，并进入调试模式，可以看到，按照绑定国家选项中默认选中“中国”，地区选项中默认选中“北京”，我们再依次选中国家多选框组中的“美国”、“巴西”和“英国”，并在地区多选框中选中“上海”和“杭州”，可以看到如图 5.8 所示的效果。

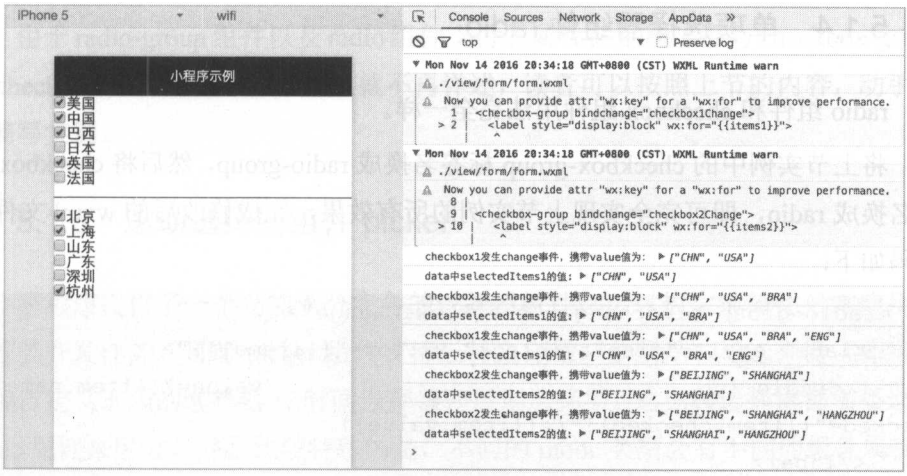


图 5.8

在每次选择一个多选框的同时，控制台的 Console 标签页里都会输出由选中项的 name 字段组成的数组，以及保存在 data 中的由已选择的项组成的数组 selectedItems1 和 selectedItems2。

在这段示例中，最重要的部分即在 checkbox 组件的属性 value 中绑定{{item.name}}数组项，其值会在该多选项被选中后通过 checkbox-group 组件的 bindchange 属性绑定的事件中显示，以事件对象的 detail.value 数组作为参数传入，因此可以在这个事件中获取已选择的多选项。

可能有读者会问，如果需要直接获取已经选择的多选框的值，应该通过哪个属性呢？

实际上可以看到 checkbox-group 组件并没有提供这样一个属性，而只是提供了一个 bindchange 属性用来绑定事件，这也是顺应了数据绑定开发方式的一种解决思路。解决方法则是像本例中一样，需要通过在 Page()的 data 对象中定义一个状态数组变量来实现，程序中其他地方如果需要访问这个组件的选中状态，则直接通过访问 data 中的状态数组即可。以本例说明，就是 this.data. selectedItems1 以及 this.data. selectedItems1。这也是数据绑定思维与传统开发思维的不同，读者将在后续的实例讲解中，进一步体会这种开发思维的转变。其实数据绑定开发的思维本身也更为清晰和直观，且能提高不少开发的效率，让开发者把精力放在业务逻辑本身。

5.1.4 单项选择器组件 radio

radio 组件和 checkbox 组件用法完全一样。

将上节实例中的 checkbox-group 标签名换成 radio-group，然后将 checkbox 标签名换成 radio，即可完全实现上节实例的所有效果，完成修改后的 wxml 文件的代码如下：

```
<radio-group bindchange="checkbox1Change">
  <label style="display:block" wx:for="{{items1}}">
    <radio value="{{item.name}}" checked="{{item.checked}}"/>{{item.value}}
  </label>
</radio-group>

<view style="margin-top:30rpx;margin-bottom:30rpx;border-top:1px solid #000"></view>

<radio-group bindchange="checkbox2Change">
  <label style="display:block" wx:for="{{items2}}">
    <radio value="{{item.name}}" checked="{{item.checked}}"/>{{item.value}}
  </label>
</radio-group>
```

js 文件保持不变，虽然 js 代码中有 checkbox1Change 和 checkbox2Change 方法，但是对组件的行为并不会有任何影响，如果读者介意也可以对这两个方法进行改名。

运行后尝试勾选不同的国家和地区，可以得到如图 5.9 的运行效果。

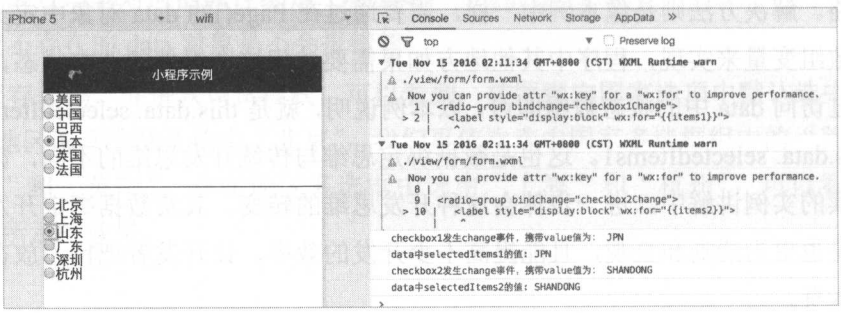


图 5.9

由于 radio-group 组件以及 radio 组件支持的属性甚至用法都与 checkbox-group 和 checkbox 组件完全一样，这里就不再详述，读者可以按照上节的内容，动手尝试编写实例。

5.1.5　滚动选择器组件 picker

小程序提供了一个功能十分完备的选择器组件，可以极大地简化日常应用中的这类开发任务。目前 picker 组件支持三种选择器，分别是普通选择器（即可以实现自定义选项的选择器）、时间选择器和日期选择器。这三种选择器都需要通过 mode 属性来区分，并且比较特殊的是，不同的 mode 类型会有不同的组件属性。

picker 组件的标签语法为：

```
    <picker range="{{['周一','周二','周三','周四','周五','周六','周日
']}}">
        点击这里选择星期
    </picker>
```

保存该代码，并点击文字区域后的效果如图 5.10 所示。

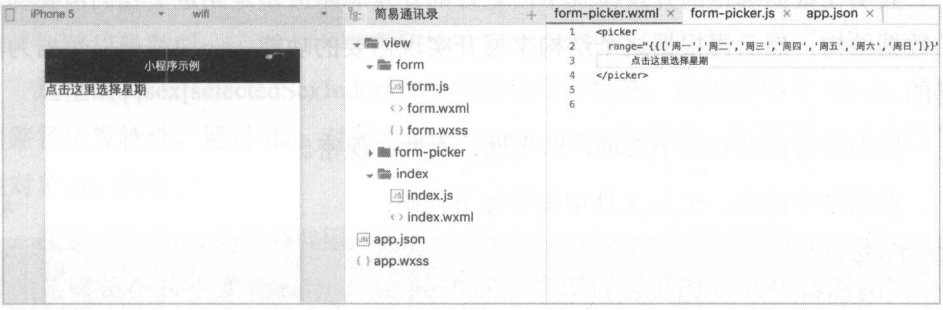


图 5.10

可以看到，我们没有在该代码中没有设置 mode 属性，最终的效果是默认为普通选择器的。另外需要注意的是，在 picker 标签中，我们需要有文字内容或者其他表单组件才能触发 picker 面板，picker 本身是不能触发 picker 选择器面板的。如果想把上述代码中的文字内容“点击这里选择星期”删掉，则 picker 是无法触发选择器面板的。

下面按照官方文档提供的在不同 mode 类型下 picker 组件的不同属性，通过一个实例来了解 picker 组件的使用方法。

当 mode 的值为默认的 selector，即为普通选择器时，picker 组件的属性如表 5.4 所示。

表 5.4　mode 属性值为 selector 时，picker 组件属性列表

属性名	类　型	默认值	说　明
range	Array	[]	mode 为 selector 时，range 有效
value	Number	0	mode 为 selector 时，是数字，表示选择了 range 中的第几个，从 0 开始
bindchange	EventHandle		value 改变时触发 change 事件，event.detail = {value: value}

表 5.4 中对 value 属性的说明不是很清晰，其实该属性是指定 picker 组件在显示选择项面板时默认选中的选择项，其值与格式根据 mode 的值不同而不同。在 mode 值为 selector 时，该值为数字，表示选择了 range 数组属性的第几个的索引值，作为数组索引，当然是会从 0 开始，如果不指定，该属性在 mode 为 selector 时默认为 0。

我们编写一个尽量精简的实例来实现性别选择器，这在我们实现用户注册功能时会非常有用。

其实在数据绑定的开发模式中，更好的开发思路应该是先定义我们需要绑定的数据结构，然后再根据这个结构来展开实现需要的功能。一切都是以数据为驱动的。

性别通常都含有三个选项，即男性、女性、保密。

根据这个需求，在 js 文件中编写如下代码：

```
Page({
  data:{
    sex:['男性','女性','保密']
  }
})
```

之后我们使用的 picker 组件就可以绑定这个 data 对象来展示选择项。

但只是展示选择项还不够，还需要获取用户选择的内容。在前面的 checkbox 组件的实例中，我们熟悉了在数据绑定思维开发方式中，组件属性状态的获取，是通过在组件绑定的 change 事件中改变 data 初始化对象的值来实现的。因此，这里还需要一个 data 对象来保存用户选择的项。从表 5.4 中可以得知，在 picker 中，bindchange 的事件对象传入的是选择的 range 数组的索引，因此我们需要定义一个

数组索引对象来获得用户的选择。因此我们继续在 js 文件中定义用户选择的数组索引 data 对象，修改后的 js 文件完整代码如下：

```
Page({
  data:{
    sex:['男性','女性','保密'],
    selectedSexIndex:0
  }
})
```

完成了初始化数据的定义，就可以在 wxml 文件中实现组件的效果，完整 wxml 文件代码如下：

```
<picker
  mode="selector"
  range="{{sex}}"
  value="0"

  bindchange="selectChange"
  >
    您选择的性别为：{{sex[selectedSexIndex]}}
</picker>
```

这里的{{sex[selectedSexIndex]}}绑定稍微有些特殊，此处应用了 wxml 的数据路径运算特性，通过 data 对象中 selectedSexIndex 索引值，选择了 data 中的数组对象 sex 的值。

在这段代码中，为了获取用户的选择，通过 bindchange 绑定了事件 selectChange，因此还需要在 js 中实现 selectChange 方法，实现后完整的 js 代码如下：

```
Page({
  data:{
    sex:['男性','女性','保密'],
    selectedSexIndex:0
  },
  selectChange:function(e){
    console.log(e)
    //通过事件对象 e.detail.value，我们可以获取到用户选择的 range 数组的第几个索引值
    this.setData({
```

```
        selectedSexIndex:e.detail.value
      })
    }
  })
```

保存代码，并进入调试模式，点击左侧工具栏的“编译”按钮，然后再点击“您选择的性别为”字样，便会出现选项选择面板，如图 5.11 所示。

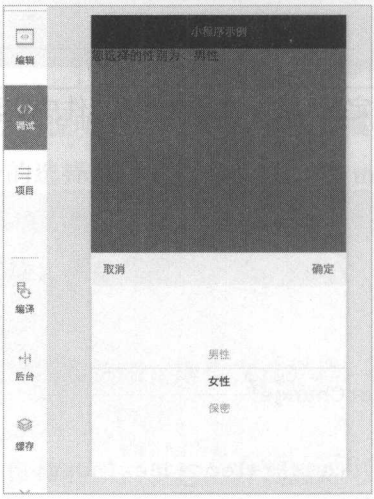


图 5.11

在该选择面板中选择“女性”选项，然后点击“确定”按钮，可以看到如图 5.12 的效果。

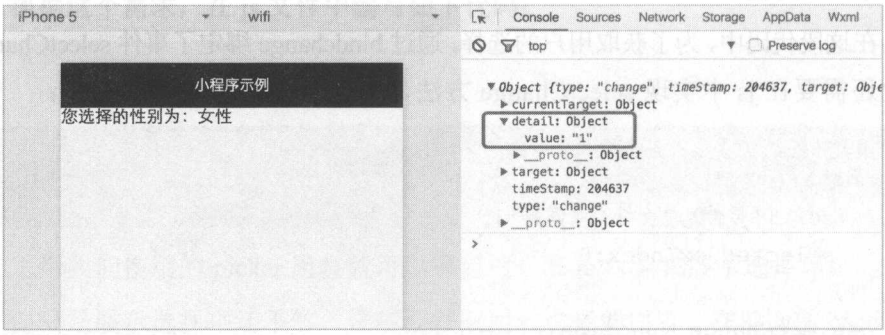


图 5.12

可以看到，通过 picker 组件很方便地实现了需要的性别选择器，获取了用户选择的内容。获取方法一样是通过 change 事件传入的事件对象 detail 来获得的。

这是在小程序中一种获取组件状态的常用做法，在类似的组件中，都需要用这种思路来实现。在后续的例子中我们还会接触到。

对于 mode 属性值为 time 时的时间选择器，官方文档提供了表 5.5 所示的说明。

表 5.5　mode 属性为 time 时的 picker 组件属性列表

属性名	类　型	默认值	说　明
value	String		表示选中的时间，格式为"hh:mm"
start	String		表示有效时间范围的开始，字符串格式为"hh:mm"
end	String		表示有效时间范围的结束，字符串格式为"hh:mm"
bindchange	EventHandle		value 改变时触发 change 事件，event.detail = {value: value}

与普通选择器对比，多了两个属性 start 和 end，表示有效时间的开始和结束，格式是 hh:mm，这样能够选择的时间只能在指定的范围内。其他属性和普通选择器的一样，只是其使用方法有所区别，value 属性在时间选择器里是用户选择的时间的字符串，格式为"hh:mm"，而不是普通选择器里的数组索引的数值。所以 bindchange 属性绑定的事件传入的事件对象 event.detail 的值也是格式为"hh:mm"的字符串。

同样，我们在实现时间选择器时，应先考虑展示的数据格式，因此我们改写 js 文件的代码为：

```
Page({
  data:{
    time:'14:12',
    timeStart:'09:01',
    timeEnd:'14:12'
  },
  selectChange:function(e){
    console.log(e)
    //通过事件对象 e.detail.value，我们可以获取到用户选择的时间字符串
    this.setData({
      time:e.detail.value
    })
  }
})
```

我们在 data 对象中定义了 time 对象，并初始化其值为'14:12'，又定义了时间范围，而把时间范围的结束与初始化的时间设置为一样，是为了更直观地体会时

间选择器 picker 组件对时间范围的控制效果。

并且我们在之前用同样的方式时实现了事件 selectChange，以及在 picker 组件的 bindchange 属性中进行了绑定。

然后在 wxml 文件中编写如下代码：

```
<picker mode="time"
        value="{{time}}"
        start="{{timeStart}}"
        end="{{timeEnd}}"

        bindchange="selectChange">
  <view class="picker">
    当前选择：{{time}}
  </view>
</picker>
```

保存代码后，进入调试模式，并点击左侧工具栏的“编译”按钮，点击视图区域中的“当前选择:”字样，即可弹出选择面板，效果如图 5.13 所示。

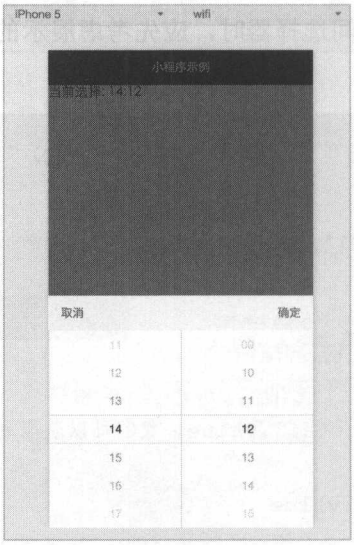


图 5.13

在该页面中，可以看到从当前选择时间是无法往后滚动的，只能往前滚动，这是因为我们在 end 属性中绑定了值为 14:12，所以只能往前滚动，这里把时间滚动到 13:05，然后点击“确定”按钮，最终的效果如图 5.14 所示。

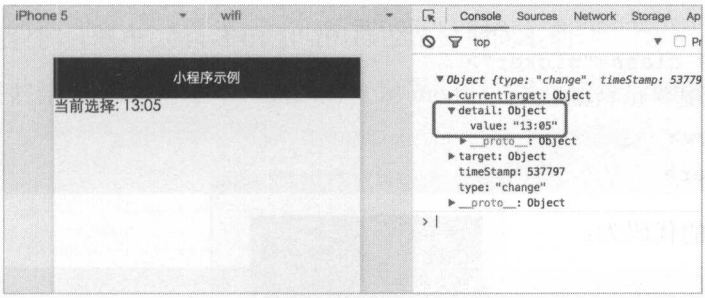


图 5.14

可见我们实现了时间选择器 picker 组件的功能，获取了用户选择的时间，同时也限制了用户可选择的时间范围。

对于 mode 属性值为 date 时的日期选择器，官方文档提供了表 5.6 所示属性说明。

表 5.6　mode 属性值为 date 时的 picker 组件属性列表

属性名	类　型	默认值	说　明
value	String	0	表示选中的日期，格式为"yyyy-MM-dd"
start	String		表示有效日期范围的开始，字符串格式为"yyyy-MM-dd"
end	String		表示有效日期范围的结束，字符串格式为"yyyy-MM-dd"
fields	String	day	有效值 year,month,day，表示选择器的粒度
bindchange	EventHandle		value 改变时触发 change 事件，event.detail = {value: value}

从表中可以看出对比 mode 值为 time 时时间选择器属性，日期选择器多了一个属性 fields，用来指定选择器的粒度，其余的属性和时间选择器的属性是一样的，包括使用方法也几乎相同，唯一区别就是 value、start 和 end 属性的字符串值格式变成了"yyyy-MM-dd"的年月日格式。

因此在实现上，我们只需在时间选择器的基础上增加一个 data 对象：fields:'day'，其他的实现思路和时间选择器是一样的，只是一些变量的初始化需要改成"yyyy-MM-dd"的日期格式，这里我们直接贴出代码。

wxml 文件的代码为：

```
<picker mode="date"
        value="{{date}}"
        start="{{dateStart}}"
        end="{{dateEnd}}"
        fields="{{fields}}"
```

```
        bindchange="selectChange">
    <view class="picker">
      当前选择: {{date}}
    </view>
</picker>
```

js 文件的代码为：

```
Page({
  data:{
    date:'2016-10-01',
    dateStart:'2014-8-01',
    dateEnd:'2016-12-5',
    fields:'day'
  },
  selectChange:function(e){
    console.log(e)
    //通过事件对象 e.detail.value，我们可以获取到用户选择的日期字符串
    this.setData({
      date:e.detail.value
    })
  }
})
```

保存运行后的效果如图 5.15 所示。

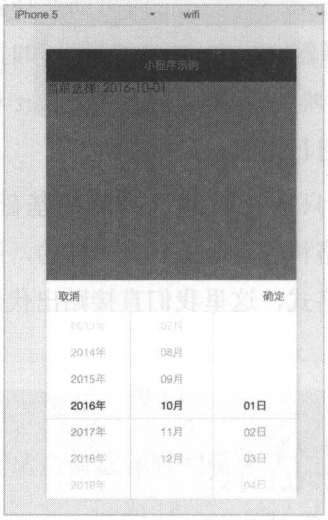


图 5.15

在这里我们只能在 2014-8-01 和 2016-12-5 日期间滚动。

如果我们修改 data 对象中 fields 的值为 month，保存运行后的效果如图 5.16 所示。

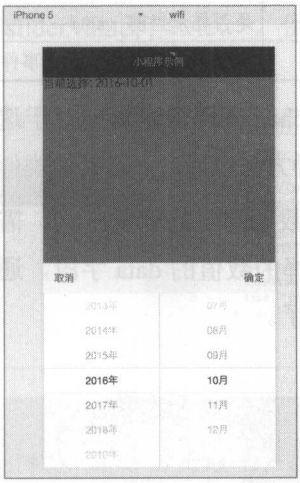


图 5.16

可以看出，这里只能选择年份和月份。

修改 data 对象中 fields 变量的值为 year 的效果，读者可以自己动手尝试一下。

5.1.6　滑动选择器组件 slider

滑动选择组件 slider 用来实现一个数值选择的滑块，对于移动端设备而言，比起使用一个输入框，然后调出输入法让用户输入数值，要方便很多，不但交互体验更好，也能节省用户的操作时间，更能降低操作的难度。

slider 组件的标签语法为：

```
<slider />
```

对其可用的属性，官方文档提供了如 5.7 所示的列表。

表 5.7　slider 组件可用的属性列表

属性名	类　型	默认值	说　明
min	Number	0	最小值
max	Number	100	最大值
step	Number	1	步长，取值必须大于 0，并且可被(max - min)整除

续表

属性名	类　型	默认值	说　明
disabled	Boolean	false	是否禁用
value	Number	0	当前取值
show-value	Boolean	false	是否显示当前 value
bindchange	EventHandle		完成一次拖动后触发的事件，event.detail = {value: value}

这里又一次见到了 bindchange 属性，可见对于选择输入类表单组件，其处理方式都是一样的，学会了这种方法以后，学习其他类似的组件就都会觉得非常轻松。

和前面一样，这种基于数据绑定的开发方式，需要先定义与组件属性对应的 data 字段，以及保存用户选择的数值的 data 字段，通过前几个例子的讲解，读者应该也能有编写代码的思路了。

js 文件的完整代码为：

```
Page({
    data:{
        initValue:0,
        sStep:5,
        sMin:0,
        sMax:70,

        sliderValue:0,
        isShowValue:true
    },
    sliderChanged:function(e){
        //通过事件对象 e.detail.value，我们可以获取到用户选择的数值
        console.log(e)
        this.setData({
            sliderValue:e.detail.value
        })
    }
})
```

在 wxml 文件中编写如下代码：

```
<slider
    value="{{initValue}}"
    step="{{sStep}}"
    show-value="{{isShowValue}}"
    min="{{sMin}}"
```

```
    max="{{sMax}}"

    bindchange="sliderChanged"
    />
```

<view>当前滑块的值:{{sliderValue}}</view>，保存并进入调试模式，最终效果如图 5.17 所示。

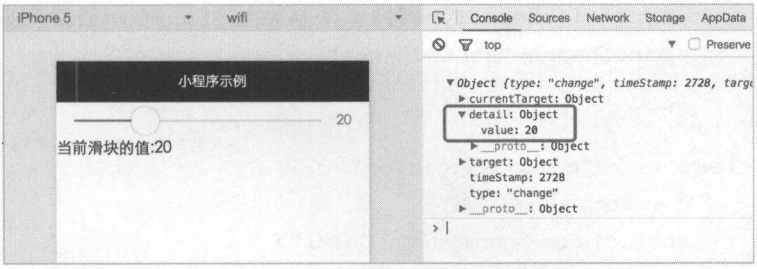


图 5.17

有一点需要注意，滑块组件的 change 事件是在用户滑动动作结束后才触发，在 data 对象中定义的字段 sliderValue 的值才会被 sliderChanged 事件更新。

5.1.7　开关选择器组件 switch

模仿 iOS 开关滑块的开关选择组件 switch 也是现在移动端开发中非常常用的一种效果，其操作相对于复选框而言更加直观。小程序提供的 switch 组件支持 iOS 的开关样式和传统的复选框样式，可以在开发中根据使用场景来进行选择。

其标签语法是：

```
<switch />
```

官方文档提供的可用属性如表 5.8 所示。

表 5.8　switch 组件的可用属性列表

属性名	类　型	默认值	说　明
checked	Boolean	false	是否选中
type	String	switch	样式，有效值：switch, checkbox
bindchange	EventHandle		checked 改变时触发 change 事件，event.detail={ value:checked}

下面结合之前介绍过的 label、button、picker 组件，来实现稍微复杂一点的 switch 组件的例子，并通过几个简单的 css 属性，将页面稍微美化一点。读者尽管

放心，通过之前的学习，这个实例理解起来依然非常轻松。

在 wxml 文件中编写如下代码：

```
<view>
    <view class="commonMarginTop">
        <switch
            checked="{{isChecked}}"
            type="{{chosedType}}"
            bindchange="stateChange"
        />
    </view>
    <view class="commonMarginTop">
        当前 switch 组件的状态：
        <label class="commonFontColor">
            {{switchState?'开':'关'}}
        </label>
    </view>
    <view class="commonMarginTop">
        <picker style="color:#999" range="{{sType}}"
            bindchange="changeSwitchType"
            >
            <button size="mini">
            点击选择 switch 组件的类型
            </button>

        </picker>
    </view>
</view>
```

代码中，我们使用 view 组件将页面分割成了三个区域，第一个 view 区域放置了 switch 标签，所有的属性都绑定了数据，包括用 bindchange 属性绑定了 stateChange 方法；第二个 view 区域用来放置状态的说明，其中又用到了 label 标签，但是 label 标签在这里的用途仅仅是作为内联标签，并没有用到其事件关联的特性，并且在绑定时使用了 wxml 数据绑定的三元运算符特性（也叫问号表达式），即对表达式“switchState?'开':'关'”做了求值，当 switchState 值为 true 时，返回字符串“开”，否则返回字符串“关”；第三个 view 区域放置了一个滚动选择器 picker 组件，其选择项是 switch 组件的 type 属性值，用来改变切换开关的样式风格，并

且 picker 组件的触发元素是一个 button 组件，并且这个 button 组件的尺寸属性 size 值为 mini。

在这基础之上，为了基本的美观，我们又引入了两个简单的样式类，一个通过 class="commonMarginTop"分别应用在每个 view 标签里，另一个放在 label 里区分了“关”字的颜色。

因此还需要在 wxss 文件里编写如下代码：

```
.commonMarginTop {
    margin-top:50rpx;
    text-align: center;
}

.commonFontColor {
    color:red
}
```

保存以后，即可在视图区域看到如图 5.18 的页面效果。

图 5.18

可以看到页面的组件有了一定的间距，且排版更加美观。

我们继续在 js 文件中编写如下代码：

```
Page({
    data:{
        isChecked:false,
```

```
        chosedType:'switch',
        sType:['switch','checkbox'],
        switchState:false,
    },
  stateChange:function(e){
                    //该事件绑定在 switch 组件上
                    //通过改变绑定在 data 上的字段 switchState 的值，来记
录组件的开关状态
                    //并通过绑定在 label 标签上的三元运算符按照开关状态显
示"开"和"关"字样
        console.log(e)
        this.setData({
            switchState:e.detail.value
        })
        console.log('switchState:',this.data.switchState)
    },
    changeSwitchType:function(e){
        this.setData({
            chosedType:this.data.sType[e.detail.value]
        })
    }
})
```

通过在 wxml 文件中的数据绑定，其逻辑代码就变得非常简单和直观了。

这里只是在 data 中绑定了初始化的数据，并通过绑定的事件 stateChange 和 changeSwitchType 来改变相应的绑定字段的初始值。

stateChange 事件被绑定在 switch 组件的 bindchange 属性上，通过传入的事件对象的 e.detail.value 值，可以获得 switch 组件当前的状态，这里用 setData()方法更新了 switchState 的值，在 label 标签中用三元操作符绑定的{{switchState?'开':'关'}}就可以通过绑定的 switchState 来显示“开”和“关”字样。

changeSwitchType 事件被绑定在 picker 组件的 bindchange 属性上，通过传入的 e.detail.value 值，可以获取到用户选择了第几个值的索引，因此通过 sType[e.detail.value] 即可得到用户选择的开关组件的样式。

保存代码，进入调试模式，并点击“点击选择 switch 组件的类型”按钮，效果如图 5.19 所示。

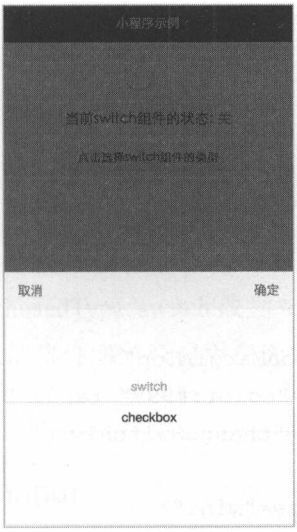


图 5.19

在滚动选择器中选择 checkbox 样式，然后点击“确定”按钮，运行效果如图 5.20 所示。

图 5.20

这里可见 switch 组件的样式变成了复选框的样式。

我们在 js 代码中取得了数组 sType 的值，实际上也可以通过 wxml 的数组运算特性，把这个运算放在 wxml 文件里实现。

将 js 文件的代码修改为：

```
<view>
    <view class="commonMarginTop">
        <switch
            checked="{{isChecked}}"
            type="{{sType[chosedIndex]}}"
```

```
            bindchange="stateChange"
        />
    </view>
    <view class="commonMarginTop">
        当前 switch 组件的状态:
        <label class="commonFontColor">
            {{switchState?'开':'关'}}
        </label>
    </view>
    <view class="commonMarginTop">
        <picker style="color:#999" range="{{sType}}"
            bindchange="changeSwitchType"
            >
            <button size="mini">
            点击选择 switch 组件的类型
            </button>

        </picker>
    </view>

</view>
```

可以看到我们将原来代码中绑定的 type="{{chosedType}}"改为了 type="{{sType[chosedIndex]}}"，直接在 wxml 里进行了数组的取值。

相应地，我们修改 js 的代码为：

```
Page({
    data:{
        isChecked:false,
        sType:['switch','checkbox'],
        switchState:false,
        chosedIndex:0
    },
    stateChange:function(e){
        console.log(e)
        this.setData({
            switchState:e.detail.value
        })
        console.log('switchState:',this.data.switchState)
```

```
    },
    changeSwitchType:function(e){
        this.setData({
            chosedIndex:e.detail.value
        })
    }
})
```

其中把原来代码中的 chosedType:'switch'改为 chosedIndex:0，这样只需通过 changeSwitchType 事件传入的事件对象获得用户选择的类型数组索引，并同步到 chosedIndex 里即可。

5.1.8　输入框组件 input

输入框是应用当中最常用，也是最重要的接受用户输入的表单组件。

小程序中的 input 组件提供了输入框所需的所有功能，且比 HTML5 提供的 input 标签更为强大易用。其标签语法为：

```
<input />
```

官方提供 input 组件支持的所有属性，如表 5.9 所示。

表 5.9　input 组件支持的属性列表

属性名	类　型	默认值	说　明
value	String		输入框的初始内容
type	String	text	input 的类型，有效值：text, number, idcard, digit, time, date
password	Boolean	false	是否是密码类型
placeholder	String		输入框为空时占位符
placeholder-style	String		指定 placeholder 的样式
placeholder-class	String	input-placeholder	指定 placeholder 的样式类
disabled	Boolean	false	是否禁用
maxlength	Number	140	最大输入长度，设置为-1 的时候不限制最大长度
auto-focus	Boolean	false	自动聚焦，拉起键盘。页面中只能有一个<input/>或<textarea/>设置 auto-focus 属性

续表

属性名	类　型	默认值	说　明
focus	Boolean	false	获取焦点（开发工具暂不支持）
bindinput	EventHandle		除了 date/time 类型外的输入框，当键盘输入时，触发 input 事件，event.detail = {value: value}，处理函数可以直接 return 一个字符串，将替换输入框的内容
bindfocus	EventHandle		输入框聚焦时触发，event.detail = {value: value}
bindblur	EventHandle		输入框失去焦点时触发，event.detail = {value: value}

可以看到小程序为我们提供了非常丰富的功能。

通过之前实例的讲解，相信读者对这些属性如何使用已经比较清楚了。这里通过综合应用之前的组件，来控制 input 组件的样式，体会 input 组件每个属性的用途。

在 wxml 文件中编写如下代码：

```
<view>
    <view class="section">
        input 输入框
        <input
                class="inputStyle"
                value="{{initValue}}"
                type="{{iTypes[pickedTypeIndex]}}"
                password="{{isPassword}}"
                placeholder="{{placeholderText}}"
                placeholder-class="placeholderStyle"
                disabled="{{isDisabled}}"
                maxlength="{{maxlength}}"
                auto-focus="{{isAutoFocus}}"
                focus="{{isFocus}}"

                bindinput="inputKeypressEvent"
                bindfocus="inputFocusEvent"
                bindblur="inputBlurEvent"
        />
    </view>
    <view class="section">
```

```
        <picker
            value="0"
            range="{{iTypes}}"
            bindchange="pickerChangeEvent">
            <button size="mini">点击选择输入框样式</button>
        </picker>
    </view>
    <view class="section">
        <label for="isPassword">是否为密码</label>
        <switch id="isPassword" class="switchStyle"
            bindchange="switchPwdChangeEvent" />
    </view>
    <view class="section">
        <label for="isIsDisabled">是否禁用</label>
        <switch id="isIsDisabled" class="switchStyle"
            bindchange="switchDisChangeEvent" />
    </view>
    <view class="section">
        最大输入字符数
        <slider min="0" max="20" show-value
            value="20"
            bindchange="sliderChangeEvent" />
    </view>
</view>
```

通过之前的介绍，这一段代码比较容易理解，这里就不再详细讲解。需要注意的是 label 标签的使用，对于这个例子，我们使用 for 属性关联了 switch 组件，因此，点击 switch 组件前面的文字内容，即可触发 switch 组件的切换事件。

这里引入了一些样式，因此在 wxss 文件中，编写如下样式代码：

```
.section{
    margin-top:70rpx;
    text-align: center;
}

.inputStyle{
    width:95%;
    height:70rpx;
    margin:auto;
```

```
    margin-top:20rpx;
    border:1px solid #CCCCCC;
    border-radius: 5px;
    text-align: left;
}

.switchStyle{
    vertical-align:middle;
    padding-left:50rpx;
}
```

从这些样式中可以看出，小程序的 wxss 语法和 css 是完全一样的。

接着在 js 文件中编写如下代码：

```
Page({
    data:{
        initValue:'初始化的文字内容',
        iTypes:["text", "number", "idcard", "digit", "time",
"date"],
        isPassword:false,
        placeholderText:"请输入文字",
        isDisabled:false,
        maxlength:20,
        isAutoFocus:false,
        isFocus:false,
        pickedTypeIndex:0,
    },

    /**
     * input 组件的输入事件
     */
    inputKeypressEvent:function(e){
        console.log('输入了内容，事件对象为:',e.detail)
    },

    /**
     * input 组件的获得焦点事件
     */
    inputFocusEvent:function(){
```

```
    console.log('获取了焦点')
},

/**
 * input 组件的失去焦点事件
 */
inputBlurEvent:function(){
    console.log('失去了焦点')
},

/**
 * picker 选择器组件的 change 事件
 */
pickerChangeEvent:function(e){
    this.setData({
        pickedTypeIndex:e.detail.value
    })
},

/**
 * 控制以密码方式显示的 switch 开关组件的 change 事件
 */
switchPwdChangeEvent:function(e){
    this.setData({
        isPassword:e.detail.value
    })
},

/**
 * 控制是否禁用 input 组件的 swtich 开关组件的 change 事件
 */
switchDisChangeEvent:function(e){
    this.setData({
        isDisabled:e.detail.value
    })
},

/**
 * 设置 input 组件最大输入字符数
```

```
        */
        sliderChangeEvent:function(e){
            this.setData({
                maxlength:e.detail.value
            })
        }
    })
```

这些实现方法在本章也做了多次介绍，这里不再赘述，读者根据注释动手尝试即可。

保存后进入调试模式，执行效果如图 5.21 所示。

这基本上已经涵盖了 input 组件的大部分属性，读者可以一一尝试。

需要注意的是，这里的输入框类型在模拟器上是看不到效果的，有条件的读者可以尝试在手机上运行程序，如图 5.22 所示是将输入框类型改为数字，并删除输入框里的默认文字后，弹出的输入法面板效果。

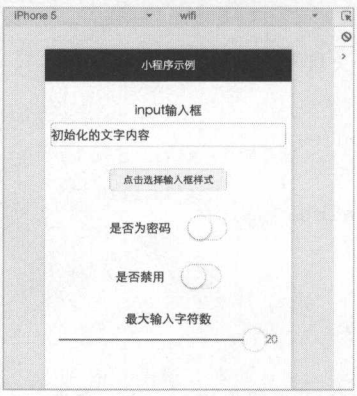


图 5.21

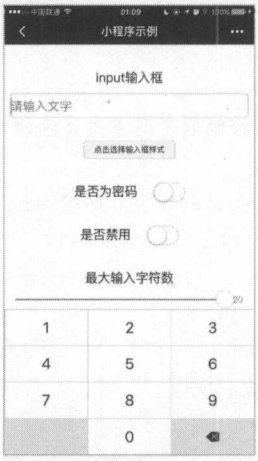


图 5.22

需要读者注意的是，bindinput 事件的 e.detail 对象比较特殊，其 detail 参数不但会传入在 input 组件文本框里键入的文字内容 value，还会附带一个 cursor 参数，其类型是数值，顾名思义，是输入文字的当前光标位置，例如我们在本例中的“初始化的”之后输入文字 te，然后控制台输出的 console.log 日志内容如图 5.23 所示。

图 5.23

本例可以说将之前讲过的组件都串讲了一遍，相信读者通过本章节练习即可熟练掌握。

5.1.9　多行输入框组件 textarea

多行输入框 textarea 组件用来处理大段的文字输入，应用场景也非常广泛。如 QQ 个人说明、微信备注等场景都需要使用 texterea 组件来实现。

其标签语法为：

```
<textarea />
```

官方文档提供的属性如表 5.10 所示。

表 5.10　textarea 组件支持的属性列表

属性名	类　型	默认值	说　明
value	String		输入框的内容
placeholder	String		输入框为空时占位符
placeholder-style	String		指定 placeholder 的样式
placeholder-class	String	textarea-placeholder	指定 placeholder 的样式类
disabled	Boolean	false	是否禁用
maxlength	Number	140	最大输入长度，设置为 0 的时候不限制最大长度
auto-focus	Boolean	false	自动聚焦，拉起键盘。页面中只能有一个<textarea/>或<input/>设置 auto-focus 属性

续表

属性名	类　型	默认值	说　明
focus	Boolean	false	获取焦点（开发工具暂不支持）
auto-height	Boolean	false	是否自动增高，设置 auto-height 时，style.height 不生效
bindfocus	EventHandle		输入框聚焦时触发，event.detail = {value: value}
bindblur	EventHandle		输入框失去焦点时触发，event.detail = {value: value}
bindlinechange	EventHandle		输入框行数变化时调用，event.detail = {height: 0, heightRpx: 0, lineCount: 0}

该表中的属性绝大多数都和 input 是一样的，使用方法我们也已经熟悉。

只有 auto-height 属性和 bindlinechange 事件比较特殊，因此我们就通过一个简单的例子来了解这两个属性。

在 wxmal 文件中编写如下代码：

```
<textarea
    style="border:1px solid #CCCCCC;"
    auto-height
    bindlinechange="textareaChangeEvent" />
```

接着在 js 文件中编写如下代码：

```
Page({
    textareaChangeEvent:function(e){
        console.log(e)
    }
})
```

这里只是在绑定的 textareaChangeEvent 中用 console.log 打印了事件对象 e，保存并进入调试模式，然后录入内容并按回车键，效果如图 5.24 所示。

这里有一个没有什么影响但是并未按预期的执行结果，即小程序运行后，会首先触发一次 bindlinechange 事件，然后换行时会再触发一次，因此在控制台中便有了两次换行事件触发的 console.log 输出的日志，这里需要注意的是图 5.24 所示方框标注的 detail 对象的数据内容。

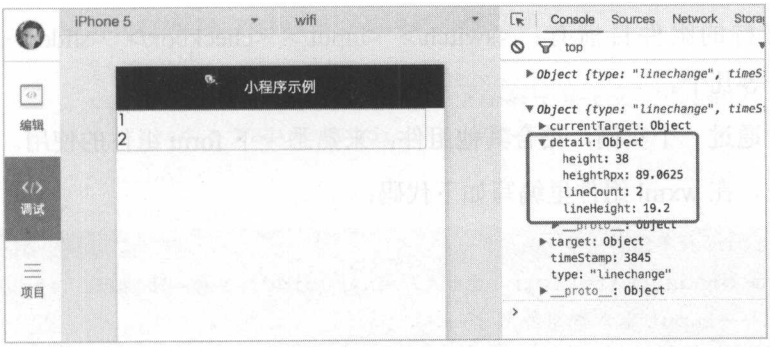


图 5.24

其中 height 值是整个 textarea 文本框以 px 单位计算的高度，即 38，heightRpx 则是以 rpx 单位计算的高度值为 89.0625，lineCount：2 是指总行数是 2 行，lineHeight：19.2 则就是每行的高度是 19.2px，这里没有继续给出 rpx 的高度是因为只要我们有了之前总高度的换算关系，即可根据换算系数计算出 lineHeight 的 rpx 单位尺寸。读者有兴趣可以根据前面章节所述内容自行计算一下。

5.1.10　表单组件 form

有前端开发经验的读者肯定不会对 html 里的表单标签 form 陌生，但是小程序提供的 form 用途和 html 里的完全不同，因为这里的 form 不是用来做 post 或者 get 提交的，而是通过事件对象来获取 form 表单内的组件数据。熟悉了前面内容的读者肯定也想到了，是通过事件的 event.detail 事件来获取。

使用该组件需要配合 button 组件的 form-type 属性来使用，如果指定为 submit，则点击按钮后会触发 form 组件的 bindsubmit 事件；如果 button 组件的 form-type 属性指定为 reset，则点击 button 按钮后会触发 form 组件的 bindreset 事件。

官方提供的属性如列表 5.11 所示。

表 5.11　form 组件支持的属性列表

属性名	类　型	说　明
report-submit	Boolean	是否返回 formId 用于发送模板消息
bindsubmit	EventHandle	携带 form 中的数据触发 submit 事件，event.detail = {value : {'name': 'value'} , formId: ''}
bindreset	EventHandle	表单重置时会触发 reset 事件

其支持的组件目前有：<switch/> <input/> <checkbox/> <slider/> <radio/> <picker/>等几个。

下面通过一个实例，结合其他组件，来熟悉一下 form 组件的使用。

首先，在 wxml 组件里编写如下代码：

```
<!--form 表单组件-->
<form bindsubmit="formSubmitEvent" bindreset="formResetEvent">
    <!--input 输入框组件部分-->
    <view class="top">
        <input name="theInputValue" placeholder="在这里输入的内会被
form 表单捕获" auto-focus/>
    </view>
    <!--switch 开关组件-->
    <view class="top">
        <switch name="theSwitch" />
    </view>
    <!--checkbox 复选框组件-->
    <view class="top">
        <checkbox-group name="theCheckboxValue">
            <label style="display:block">
                <checkbox value="checkbox1" />
                复选框 1
            </label>
            <label style="display:block;margin-top:20rpx">
                <checkbox value="checkbox2" />
                复选框 2
            </label>

        </checkbox-group>
    </view>
    <!--slider 滑动选择组件-->
    <view class="top">
        <slider name="theSliderValue" show-value />
    </view>
    <!--radio 单选框组件-->
    <view class="top">
        <radio-group name="radioValue">
            <label style="display:block">
```

```
                <radio value="radio1"/>单选框 1
            </label>
            <label style="display:block;margin-top:20rpx">
                <radio value="radio2"/>单选框 2
            </label>
        </radio-group>
    </view>
    <!--picker 选择器组件-->
    <view class="top">
        <picker name="thePickerValue" range="{{['红色','蓝色','绿色
','橙色','金色']}}">
            <button size="mini">点击这里弹出 picker 选择器</button>
        </picker>
    </view>
    <!--textarea 多行文本输入框组建-->
    <view class="top">
        <textarea name="placeHolderValue"
                  placeholder="这里可以输入多行文本"
                  style="border:1px solid #CCCCCC;width:90%; margin:
auto"
                   />
    </view>
    <!--submit 类型的 button 按钮组件-->
    <view class="top">
        <button  form-type="submit" >提交</button>
    </view>
    <!--reset 类型的 button 按钮组件-->
    <view class="top">
        <button  form-type="reset" >重置</button>
    </view>

</form>
```

在这段代码中，我们在 form 组件中使用了本章所学的输入组件。如果有较多的表单需要用户输入，特别是像用户注册这样的场景，如果让每个表单组件都使用 change 事件来获得用户的输入状态，则代码会变得比较复杂，也更难以维护，form 的主要作用则是可以为我们简化在这种场景下的代码组织，并简化开发。

需要注意的是，在使用 form 组件时，其内部组件是通过 name 属性来进行标

识，在通过 bindsubmit 事件传入 detail.value 对象时，是通过将 name 属性名作为键值名称传入的，比如本例中 name 为 theInputValue 的 input 组件，假设用户没有输入，则会在 detail.value 中传入{theInputValue:" "}。也就是说，如果在标签里没有指定 name 属性的名称，那么使用 form 是无法获得用户输入的。

这里还需要注意的是，对于 checkbox 组件和 radio 组件的使用，是必须配套 checkbox-group 和 radio-group 来使用，事实上这也是 checkbox 和 radio 的正确使用方式，即不能单独使用。另外 checkbox-group 组件在 change 事件中传入的 detail.value 对象是由其复选框组件 checkbox 的 value 值组成的数组，而 radio-group 组件在 change 事件中传入的 detail.value 对象则是 radio 组件的 value 值字符串，这和多选框及单选框的特点也是相对应的。

在 form 组件中，我们用 view 组件来区分不同的输入组件区域，并且稍稍优化了一下界面。因此，在 wxss 文件中编写简单的样式代码如下：

```
.top{
    margin-top:50rpx;
    text-align: center;
}
```

这两行样式对读者而言应该非常简单，这里不再赘述。

实现完页面部分，需要继续实现以下绑定的事件行为，继续在 js 文件中编写如下代码：

```
Page({
    /**
     * 在 form 组件 formSubmitEvent 属性中绑定的事件
     */
    formSubmitEvent:function(event){
        console.log(event)
    },
    /**
     * 在 form 组件 formResetEvent 属性中绑定的事件
     */
    formResetEvent:function(event){
        console.log(event)
    }
})
```

可以看到在这段代码中，我们只有 formSubmitEvent 和 formResetEvent 两个事件的实现，代码量非常少。

保存代码，并进入调试模式，为了便于看到点击提交和重置按钮的效果，依次做如下操作：

（1）在 input 输入框输入“input 输入框内容”；

（2）打开开关选择器；

（3）选择“复选框 1”和“复选框 2”；

（4）将滑动选择器滑动到 63；

（5）选择“单选框 2”；

（6）点击“点击这里弹出 picker 选择器”按钮，并选择“绿色”，确定；

（7）在多行文本框中输入“这里是多行文本框内容”；

（8）点击“提交”按钮。

完成后，程序的运行效果如图 5.25 所示。

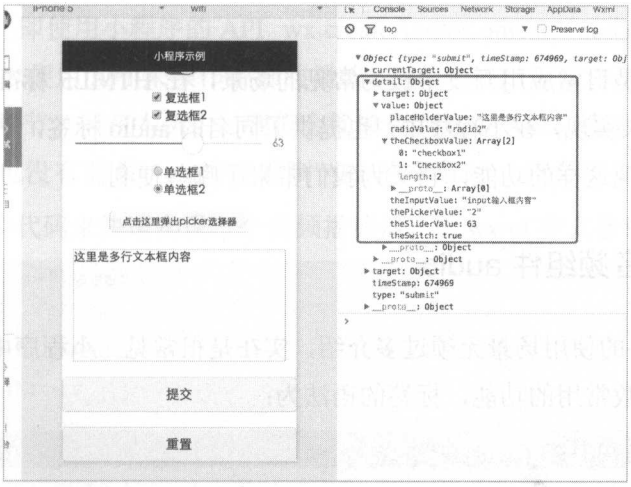


图 5.25

可以看到，通过 form 组件，虽然并没有在每个表单组件中绑定和实现 change 事件，但依然通过 bindsubmit 事件获得了所有组件的输入内容及操作状态，所有的数据都被保存在 detail.value 对象中。

接着，点击“重置”按钮，操作的结果如图 5.26 所示。

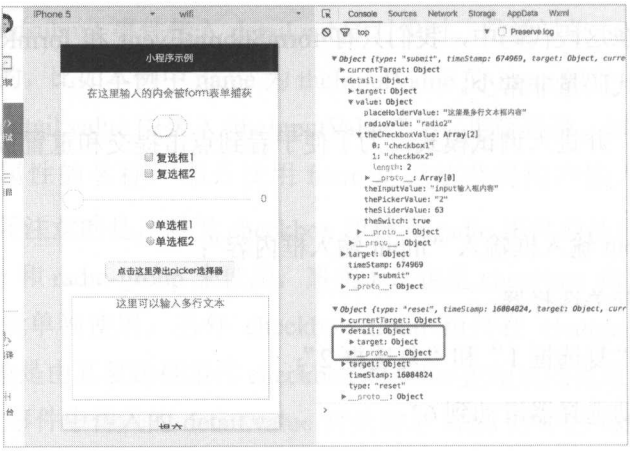


图 5.26

可以看到输入的所有内容都被清空，并且我们得到的 event 对象的 detail 中并没有包含 value 对象，符合重置功能的应用。

5.2 媒体组件

音频播放是日常应用开发中极为常见的场景，在 HTML5 标准里为我们提供了 audio 标签来实现，在小程序中，也提供了同名的 audio 标签，下面来看看在小程序中如何实现这样的功能，它又为我们带来了哪些便利。

5.2.1 音频组件 audio

audio 组件的使用场景无须过多介绍，实在是很常见。小程序中的 audio 标签封装了绝大多数常用的功能，标签的语法为：

```
<audio></audio>
```

而且其属性也非常直观，使用起来非常方便，官方提供的属性列表如表 5.12 所示。

表 5.12 audio 组件支持的属性列表

属性名	类 型	默认值	说 明
id	String		video 组件的唯一标识符
src	String		要播放音频的资源地址

续表

属性名	类　型	默认值	说　明
loop	Boolean	false	是否循环播放
controls	Boolean	true	是否显示默认控件
poster	String		默认控件上的音频封面的图片资源地址，如果 controls 属性值为 false 则设置 poster 无效
name	String	未知音频	默认控件上的音频名字，如果 controls 属性值为 false 则设置 name 无效
author	String	未知作者	默认控件上的作者名字，如果 controls 属性值为 false 则设置 author 无效
binderror	EventHandle		当发生错误时触发 error 事件，detail = {errMsg: MediaError.code}
bindplay	EventHandle		当开始/继续播放时触发 play 事件
bindpause	EventHandle		当暂停播放时触发 pause 事件
bindtimeupdate	EventHandle		当播放进度改变时触发 timeupdate 事件，detail = {currentTime, duration}
bindended	EventHandle		当播放到末尾时触发 ended 事件

表中的 id 属性其实所有的组件都可以使用，但是在 audio 组件中有个较为特殊的使用场景，即使用小程序的 API wx.createAudioContext（audioId）时，获得 audio 上下文，以便用 API 来操作音频的播放、暂停、快进等操作。虽然我们还未讲解到 API 的章节，但在本例中，我们会简单用到这个 API，读者可以先感受一下，不太理解也没有关系，后续在讲到 API 的章节时，还会再详细介绍。

（1）以最少代码来快速实现一个音频播放器，在 wxml 中直接编写如下代码：

```
<view class="base">
    <audio
        id="theAudio"
        controls
src="http://5.1015600.com/1/2014/ring/000/118/28b0e17cfab0136677648b
39cb8b7fbc.mp3"
        binderror="errFun"
        bindplay="playFun"
        bindpause="pauseFun"
        bindtimeupdate="timeUpdateFun"
        bindended="endedFun"
        ></audio>
```

```
</view>
```

（2）为了基本的界面美观，在 wxss 文件中编写如下代码：

```
.base{
    text-align: center;
    margin-top:40rpx;
}
```

（3）然后在 js 中编写如下的代码：

```
Page({
    playFun:function(){
        console.log('开始播放')
    },
    pauseFun:function(){
        console.log('播放暂停')
    },
    timeUpdateFun:function(event){
        console.log('时间轴更新了',event)
    },
    endedFun:function(){
        console.log('播放结束')
    },
    errFun:function(event){
        console.log('播放出错',event)
    }
})
```

（4）保存并进入调试模式，点击控制面板的“播放”按钮，然后大概 4 秒左右点击停止，其运行的效果如图 5.27 所示。

在点击播放开始和结束时，分别触发了 playFun 和 pauseFun 事件，并且在播放过程中，每秒都会触发 timeUpdateFun 事件，从中我们可以获取到当前播放的信息，从图 5.27 所示的线框区域中，我们可以得到当前播放位置时间 currentTime 以及总时长 duration。

为了触发出错事件，我们可以任意修改一下 audio 组件 src 指定的地址，迫使 audio 组件加载出错，以触发 binderror 事件，出错的效果如图 5.28 所示。

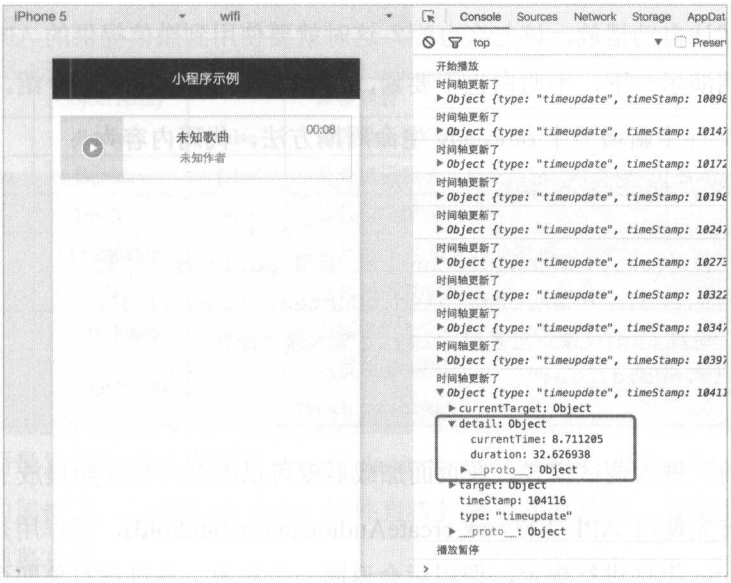


图 5.27

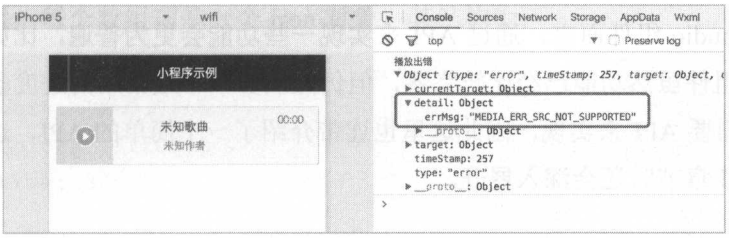


图 5.28

从图 5.28 的线框标记区中，我们可以清晰地了解到 detail 的信息内容，即 MEDIA_ERR_SRC_NOT_SUPPOERTED，对于错误码的说明，官方文档提供了如表 5.13 所示的说明。

表 5.13　错误码说明表

返回错误码	描　述
MEDIA_ERR_ABORTED	获取资源被用户禁止
MEDIA_ERR_NETWORD	网络错误
MEDIA_ERR_DECODE	解码错误
MEDIA_ERR_SRC_NOT_SUPPOERTED	不合适资源

有心的读者可能会有疑惑，audio 组件的属性中没有提供 autoplay 之类的属性，

如果要是默认自动播放，该怎么办呢？这时就要使用到微信提供的 API 了，这里简单用实例演示一下，不明白也不要紧，后续章节还会做详细的介绍。

在 js 文件中新增一个 onReady 生命周期方法，代码内容为 ：

```
onReady: function (e) {

    // 使用 wx.createAudioContext 获取 audio 操作对象
    this.audio = wx.createAudioContext('theAudio')
    // 通过 audio 操作对象的 play 方法来操作播放
    this.audio.play()
  },
```

保存后，进入调试模式，在页面加载后就可以听到音乐开始播放了。

这里首先使用 API 方法 wx.createAudioContext(audioId)，其作用是和参数指定 id 的 audio 组件进行绑定，调用后会返回一个对象，通过该对象即可操作绑定的 audio 组件。此例中，我们调用了 play()方法来进行播放。

对于 audio 组件而言，通过 API 来实现一些功能会更为普遍，比如目前提供的 audio 组件虽然功能已经比较完备，但仍然缺少一个最常用的进度条，实现这个功能就需要 API 来实现，因此这里也连带介绍了一个简单的 API，这部分内容在讲到 API 章节时还会深入展开。

5.2.2 视频组件 video

视频播放器的应用场景较为广泛，小程序的视频播放器组件 video 不但封装了常见的功能，甚至还封装了当前最为火热的弹幕功能，使得弹幕功能的实现非常方便。

video 标签的语法为：

```
<video></video>
```

官方提供的属性列表，如表 5.14 所示。

表 5.14 video 组件支持的属性列表

属性名	类 型	默认值	说 明
src	String		要播放视频的资源地址
controls	Boolean	true	是否显示默认播放控件（播放/暂停按钮、播放进度、时间）

续表

属性名	类　型	默认值	说　明
danmu-list	Object Array		弹幕列表
danmu-btn	Boolean	false	是否显示弹幕按钮，只在初始化时有效，不能动态变更
enable-danmu	Boolean	false	是否展示弹幕，只在初始化时有效，不能动态变更
autoplay	Boolean	false	是否自动播放
bindplay	EventHandle		当开始/继续播放时触发 play 事件
bindpause	EventHandle		当暂停播放时触发 pause 事件
bindended	EventHandle		当播放到末尾时触发 ended 事件
binderror	EventHandle		当发生错误时触发 error 事件，event.detail = {errMsg: 'something wrong'}

这里面最为惹眼的，莫过于 enable-danmu，danmu-list，danmu-btn 这三个与弹幕相关的属性了，这意味着 video 组件直接封装了弹幕功能，使得我们可以轻易地实现弹幕功能。

这里先介绍一下 danmu-list 需要传入的 object 对象的结构。damu-list 属性接收一个数组，每个数组项是一个 json 对象，其结构为：

```
[{
    text: '弹幕的内容 ',
    color: '#ff0000',
    time: 3
}]
```

其中，text 是弹幕的文字内容，color 是弹幕文字颜色，而 time 则是弹幕触发的时间，单位是秒。目前而言，能够自定义的弹幕内容只有这三项，其他包括弹幕的文字大小和字体都无法自定义，小程序后续是否会扩展目前还不得而知。

下面通过实例来熟悉一下。在 wxml 中编写如下代码：

```
<video id="theVideo"
    style="width:100%"
    enable-danmu
    danmu-list="{{textList}}"

src="http://flv.bn.netease.com/videolib3/1605/22/auDfZ8781/HD/auDfZ8
781-mobile.mp4">
</video>
```

这里只有一个 video 标签，设置了 enable-danmu 属性为 true，并为 danmu-list 绑定了数据{{textList}}，用 src 属性指定了一个视频的 url。

接着在 js 中实现绑定的{{textList}}数据，代码如下：

```
Page({
  data:{
    textList:[
      {
        text: '小程序强大的弹幕第一单，3s',
        color: '#ff0000',
        time: 3
      },
      {
        text: '小程序的弹幕十分强大，6s',
        color: '#ff00ff',
        time: 6
      },
      {
        text: '很好看的一部动画片，6s',
        color: '#ffffff',
        time: 6
      },
      {
        text: '画面感确实不错，10s',
        color: '#ff00ff',
        time: 10
      }
    ]
  }
})
```

textList 数组中，我们实现了四个弹幕项，其中第 6 秒则有两个弹幕。

保存代码后，直接点击播放控制条上的播放按钮，即可看到播放的效果，如图 5.29 所示。

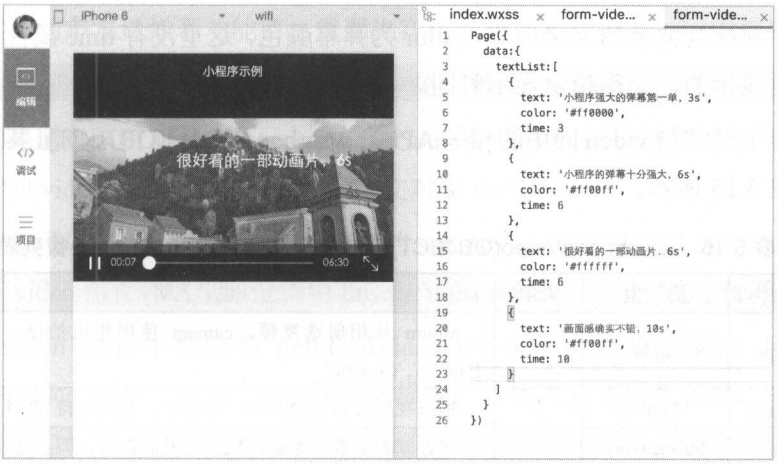


图 5.29

可以看到，这几行简短的代码即实现了非常强大的弹幕效果。

笔者在开发者工具中看到的弹幕效果有一些卡顿，但是在手机上十分的流畅，最终的运行效果令人满意，完全可以在相关的开发需求中直接使用 video 组件的弹幕功能。

video 组件除了播放网络视频外，还能够播放用户设备相册里的视频，甚至获取用户摄像头拍摄的视频。然而实现这个功能需要借助相应的 API 来实现。与 audio 类似，video 组件很多重要的功能都需要通过 API 来实现，比如目前我们已经实现了固定的弹幕内容显示，但如果需要动态的发送弹幕内容来显示呢？这也是需要借助 API。

这里继续使用相关的两个 API 来实现动态的发送弹幕到视频，以及获取用户设备里的视频进行播放。

与 audio 组件的 wx.createAudioContext(audioId) 方法相对应，video 标签也有一个 wx.createVideoContext(videoId)方法，使用方法和 wx.createAudioContext (audioId) 完全一样，通过参数 videoId，可以和指定 id 的 video 组件进行绑定，得到的 videoContext 对象有一个 sendDanmu 方法，可以用来动态的发送弹幕，其语法为：

```
sendDanmu({
text:'弹幕内容',
color:'#ff00ff ',
})
```

其中 text 为弹幕的文字内容，color 为弹幕颜色，这里没有 time 属性，因为是动态即时显示的，无需指定显示时间。

另一个经常与 video 同用的相关 API 为 wx.chooseVideo(OBJECT)，其 OBJECT 参数如表 5.15 所示。

表 5.15　wx.chooseVideo(OBJECT)接口中 OBJECT 对象支持的参数列表

参　数	类　型	必填	说　明
sourceType	StringArray	否	album 从相册选视频，camera 使用相机拍摄，默认为：['album', 'camera']
maxDuration	Number	否	拍摄视频最长拍摄时间，单位秒。最长支持 60 秒
camera	StringArray	否	前置或者后置摄像头，默认为前后都有，即：['front', 'back']
success	Function	否	接口调用成功，返回视频文件的临时文件路径，详见返回参数说明

其中选取成功后会调用 success 回调方法，并传入一个对象参数，这个对象参数包含一个 tempFilePath 属性，是存放用户选择视频的临时保存位置，我们需要用这个地址来替换 video 的 src 地址，即可实现对用户选择的视频内容进行播放。

修改上述例子的代码，加入动态发送弹幕以及选取本地视频的功能。

为了简化代码，使用 form 表单来包含 input 输入框组件和发送按钮，并在增加一个按钮来实现选择本地视频，因此将 wxml 文件修改为：

```
<video id="theVideo"
    style="width:100%"
    enable-danmu
    danmu-list="{{textList}}"
    src="{{videoSrc}}">
</video>
<form bindsubmit="submitFun">
    <input
        style="border:1px solid #CCCCCC;margin:5rpx"
        name="danmuText"
        />
    <button form-type="submit">点击发送弹幕</button>
</form>
<button style="margin-top:30rpx"
bindtap="getLocalVideoFun">获取设备视频</button>
```

在实现动态发送弹幕功能前，需要先在生命周期函数 onReady 中使用：

```
this.theVideo= wx.createVideoContext('theVideo')
```

语句先通过 API 函数 wx.createVideoContext('theVideo')来获得与 id 属性值为'theVideo'的 video 组件绑定的 video 上下文对象 theVideo。

这样就可以在 form 表单组件上绑定的 submitFun 方法中通过 this.theVideo 对象来操作 video 组件，这里通过调用 this.theVideo.sendDanmu 实现了弹幕的发送。

在 button 组件中绑定的 getLocalVideoFun 事件中，通过调用 API 函数 wx.chooseVideo 实现了对用户设备视频的播放。具体的 js 完整代码为：

```
Page({
  data:{

videoSrc:'http://flv.bn.netease.com/videolib3/1605/22/auDfZ8781/HD/a
uDfZ8781-mobile.mp4',
    textList:[
      {
        text: '小程序强大的弹幕第一单，3s',
        color: '#ff0000',
        time: 3
      },
      {
        text: '小程序的弹幕十分强大，6s',
        color: '#ff00ff',
        time: 6
      },
      {
        text: '很好看的一部动画片，6s',
        color: '#ffffff',
        time: 6
      },
      {
        text: '画面感确实不错，10s',
        color: '#ff00ff',
        time: 10
      }
    ]
  },
```

```
  /**
   * 在生命周期 onReady 方法中得到 video 上下文对象
   * 这是比较通用的一种做法，以便在其他方法中对 video 进行控制
   */
  onReady:function(){
    this.theVideo=wx.createVideoContext('theVideo');
  },
  /**
   * 通过 form 表单获得 input 输入框的内容，并通过 sendDanmu 方法发送弹幕
   */
  submitFun:function(e){
    //通过 form 表单获取 input 输入框的内容
    var danmuText=e.detail.value.danmuText
    //通过调用 theVideo 对象的 sendDanmu 方法来动态发送弹幕
    this.theVideo.sendDanmu({
      text:danmuText,
      color:'#ffffff'
    })
  },
  /**
   * 通过调用 wx.chooseVideo 函数，实现播放用户设备中的视频的功能
   */
  getLocalVideoFun:function(){
    var that = this
        wx.chooseVideo({
            sourceType: ['album','camera'],
            maxDuration: 60,
            camera: ['front','back'],
            success: function(res) {
                that.setData({
                    videoSrc: res.tempFilePath
                })
            }
        })
  }
})
```

读者可以通过代码注释继续了解相关 API 的用法。

保存代码后，如果读者有 openId，就可以在手机设备上运行该代码，在输入框输入文本后点击“点击发送弹幕”按钮，就可以看到弹幕已经发送在视频中。这里需要注意的是，弹幕必须是在视频开始播放后点击，如果在视频未播放时就点击了发送弹幕，那么弹幕内容会在视频开始播放后才在屏幕上显示，最终在设备的效果如图 5.30 所示。

点击“获取设备视频”按钮后的效果如图 5.31 所示。

图 5.30

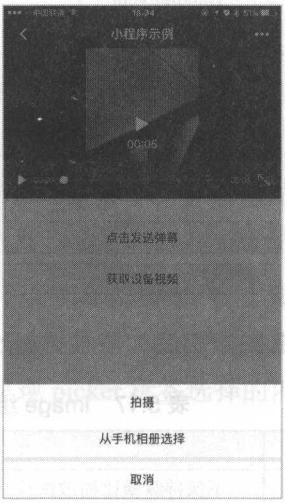


图 5.31

继续点击“拍摄”选项可以录制视频，并用 video 组件来播放；点击“从手机相册选择”选项，可以从手机相册选择现有的视频，并用 video 组件播放。

video 组件的特殊性决定了其与 API 关系更为紧密，因此这里简单介绍了两个相关 API 函数的部分用法，在后续介绍 API 的章节里会继续详细讲解。但读者仍然有必要通过练习来对小程序的 API 用法建立一些初步的认识。

5.2.3　图片组件 image

在 HTML 中，我们对 img 标签应该说是十分熟悉，在小程序中，是通过 image 组件来实现对应的功能。其使用方法非常简单直观，标签语法为：

```
<image> </image>
```

支持的属性按照官方文档提供的如表 5.16 所示。

表 5.16 image 组件支持的属性列表

属性名	类 型	默认值	说 明
src	String		图片资源地址
mode	String	'scaleToFill'	图片裁剪、缩放的模式
binderror	HandleEvent		当错误发生时，发布到 AppService 的事件名，事件对象 event.detail = {errMsg: 'something wrong'}
bindload	HandleEvent		当图片载入完毕时，发布到 AppService 的事件名，事件对象 event.detail = {height:'图片高度 px', width:'图片宽度 px'}

一共四个属性，其中最主要的就是 mode 属性的用法。

mode 属性的作用是以何种方式显示 src 属性指定的图片，在以往的 HTML 中，需要通过 css 中的 background-position 和 background-repeat 方法相结合来实现，在小程序的 image 属性中则对此作了封装，使其更加易于实现。

其一共支持 12 种显示模式，其中 3 种为缩放模式，9 种为剪裁模式，按照官方文档，其缩放模式如表 5.17 所示。

表 5.17 image 组件 mode 属性支持的缩放模式列表

模 式	说 明
scaleToFill	不保持纵横比缩放图片，使图片的宽高完全拉伸至填满 image 元素
aspectFit	保持纵横比缩放图片，使图片的长边能完全显示出来。也就是说，可以完整地将图片显示出来。
aspectFill	保持纵横比缩放图片，只保证图片的短边能完全显示出来。也就是说，图片通常只在水平或垂直方向是完整的，另一个方向将会发生截取。

而其剪裁模式如表 5.18 所示。

表 5.18 image 组件 mode 属性支持的剪裁模式列表

模 式	说 明
top	不缩放图片，只显示图片的顶部区域
bottom	不缩放图片，只显示图片的底部区域
center	不缩放图片，只显示图片的中间区域
left	不缩放图片，只显示图片的左边区域
right	不缩放图片，只显示图片的右边区域
top left	不缩放图片，只显示图片的左上边区域
top right	不缩放图片，只显示图片的右上边区域
bottom left	不缩放图片，只显示图片的左下边区域
bottom right	不缩放图片，只显示图片的右下边区域

为了更直观地认识这个属性的用法，下面通过一个实例来认识一下。

在 wxml 文件中，编写如下代码：

```
<view style="text-align:center">
    <image
        style="border:1px solid red"
         mode="{{modes[modeIndex]}}"
src="http://img02.tooopen.com/images/20160603/tooopen_sy_164101489754.jpg">

    </image>
     <picker  bindchange="pickerChangeFun"  value="{{modeIndex}}"
range="{{modes}}">
        <button>点击这里选择</button>
     </picker>
</view>
```

这段代码中，使用了一个 image 标签和一个 pick 标签，通过绑定 image 标签的 mode 属性为{{modes[modeIndex]}}，来直接呈现 picker 标签选择的内容，因此在 js 中，只需编写如下代码：

```
Page({
  data:{
    modes:[
        'scaleToFill',
        'aspectFit',
        'aspectFill',
        'top',
        'bottom',
        'center',
        'left',
        'right',
        'top left',
        'top right',
        'bottom left',
        'bottom right'],
     modeIndex:0,
  },
  pickerChangeFun:function(event){
```

```
    this.setData({
      modeIndex:event.detail.value
    })
  }
})
```

保存代码后，无需进入调试模式，就可以在视图区域看到代码运行的效果，如图 5.32 所示。

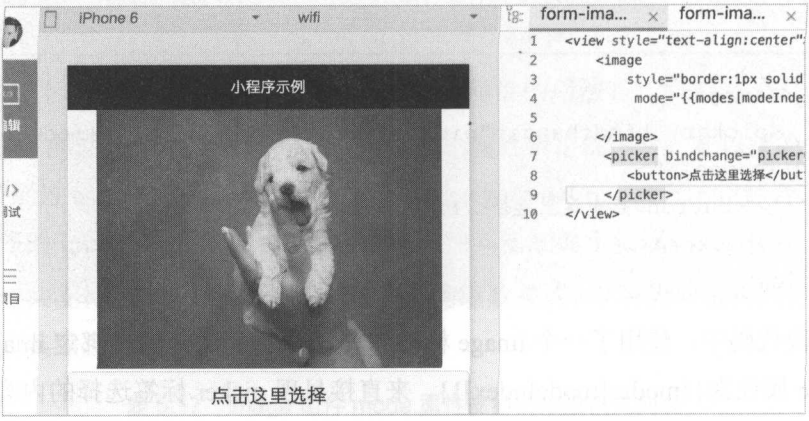


图 5.32

点击“点击这里选择”按钮，然后在弹出的选择器中，我们尝试选择“top”选项，则 image 组件会呈现如图 5.33 所示的图片剪裁区域。

可见 mode 属性为 top 值时，是直接剪裁图片的顶部区域来显示的，其他值的显示效果，读者可以自行尝试。

图 5.33

5.3　地图组件 map

地图组件在小程序中主要作用是呈现，并且我们发现该组件在开发工具中的效果并不是很完善，其呈现的效果与手机上的效果差异较大。

map 组件的标签语法为：

```
<map> </map>
```

官方提供的属性列表如表 5.19 所示。

表 5.19　map 组件支持的属性列表

属性名	类　型	默认值	说　明
longitude	Number		中心经度
latitude	Number		中心纬度
scale	Number	16	缩放级别
markers	Array		标记点
covers	Array		覆盖物

其中 markers 属性为在地图中显示标记点，即用来标记一个位置，其是一个数组，可以用来标记多个位置，我们在地图应用中搜索一个场所，如果有多个结果，则这就是多个标记点。

如果在百度地图中搜索电影院，则会有如图 5.34 所示的多个标记点。

图 5.34

这就是 markers 属性的作用，可以接受多个标记点，而每个标记点则有其固定的格式，官方提供的列表如 5.20 所示。

表 5.20　markers 属性支持的标记点参数列表

属　性	说　明	类　型	必　填	备　注
latitude	纬度	Number	是	浮点数，范围 -90～90
longitude	经度	Number	是	浮点数，范围 -180～180
name	标注点名	String	是	
desc	标注点详细描述	String	否	

对于标记点是不能够自定义图标和样式的，官方会自行提供相应的样式和结构来显示，而在实践中也发现，同样的实现在开发者工具和手机上表现出来的效果完全不同。

其中 covers 是一个覆盖物，与 markers 不同的是，其不能带有文字说明，只能是一个图标。

其用法和 markers 基本类似，也是一个数组，每个数组对象的属性见表 5.21 所示。

表 5.21　covers 属性支持的属性列表

属　性	说　明	类　型	必　填	备　注
latitude	纬度	Number	是	浮点数，范围 -90～90
longitude	经度	Number	是	浮点数，范围 -180～180
iconPath	显示的图标	String	是	项目目录下的图片路径，支持相对路径写法
rotate	旋转角度	Number	否	顺时针旋转的角度，范围 0～360，默认为 0

相比 covers 属性，markers 可以通过 iconPath 属性自定义图标，也可以自定义样式，还可以通过 rotate 属性控制旋转角度。

下面举一个简单的例子。

在 wxml 文件中编写如下代码：

```
<view style="height:1334rpx">
    <!--地图组件-->
    <map style="width:100%;height:75%"
        longitude="116.3972282409668"
        latitude="39.9086611069"
        scale="{{scale}}"
```

```
        markers="{{markers}}"
    >
    </map>
    <!-- 滑块区域，用来缩放地图-->
    <view style="margin-top:50rpx;text-align: center">
        <slider
            style="width:80%;margin:auto"
            min="1"
            max="50"
            step="3"
            value="{{scale}}"
            bindchange="changeScaleFun"
         />
    </view>
    <view style="margin-top:50rpx;text-align:center">
        缩放级别:{{scale}}
    </view>
</view>
```

在该例子中，使用了 slider 组件来控制地图的缩放，这个控件做这个操作也十分适合。

接着在 js 文件中编写如下代码：

```
Page({
  data:{
    scale:"16",
    markers: [{
      longitude:116.3975273161,
      latitude:39.9086611069,
      name: '北京天安门',
      desc: '我现在的位置'
    }]
  },
  /**
   * 绑定在 slider 组件上的 change 方法
   * 用于实现地图的缩放
   */
  changeScaleFun:function(e){
    this.setData({
```

```
        scale:e.detail.value
      })
    }
})
```

保存代码，即可在开发者工具的视图区域看到如图 5.35 所示的效果。

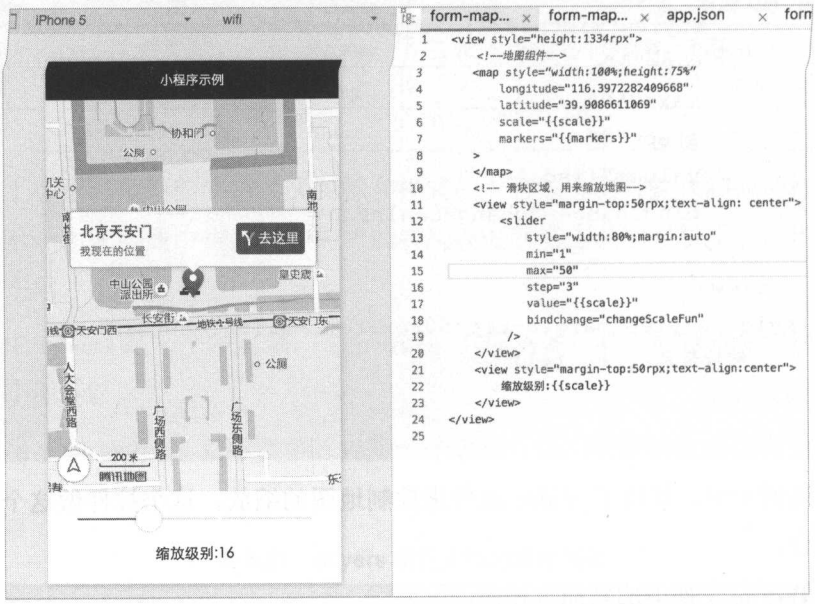


图 5.35

我们在开发工具中拖动缩放级别的 slider 组件滑块，会发现不起作用，实际上代码是正确的，这一点就是地图组件还不够完善的地方，这个效果在模拟器上无效，需要放在真机环境下操作。

从图 5.36 中可见，我们通过 slider 组件来改变缩放级别，是有效的。并且也可以从中得出，map 组件的 scale 属性值越大代表放大地图，离地面越近；值越小，代表缩小地图，离地面越远，且最大为放大 50 倍左右的级别。

map 组件更多的用途还是展示，其他的功能则需要结合 API 来实现，更深入的部分，放在介绍相关 API 的章节里继续介绍。

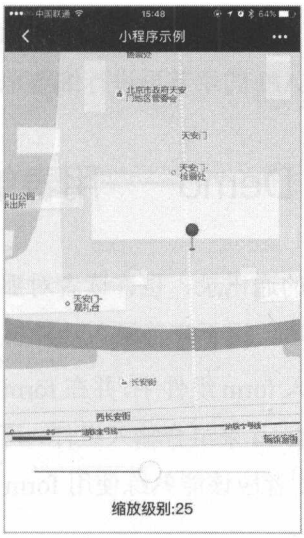


图 5.36

另外使用 map 组件需要注意，由于 map 组件本身有滑动操作，不能将其放在 scroll-view 组件中使用。

5.4　画布组件 canvas

与 HTML5 规范中的 canvas 组件类似，小程序也提供了 canvas 画布组件用来实现有关图形的功能，官方文档提供的属性列表如表 5.22 所示。

表 5.22　canvas 组件支持的属性列表

属性名	类　型	默认值	说　明
canvas-id	String		canvas 组件的唯一标识符
disable-scroll	Boolean	false	当在 canvas 中移动时，禁止屏幕滚动以及下拉刷新
bindtouchstart	EventHandle		手指触摸动作开始
bindtouchmove	EventHandle		手指触摸后移动
bindtouchend	EventHandle		手指触摸动作结束
bindtouchcancel	EventHandle		手指触摸动作被打断，如来电提醒，弹窗
binderror	EventHandle		当发生错误时触发 error 事件，detail = {errMsg: 'something wrong'}

由表 5.22 可见，除了特殊的 canvas-id 属性和 disable-scroll 属性，其他属性都是事件相关。

Canvas 画布主要的作用是使用 API 函数在其上面作图和一些其他实现，因此 canvas 的用法留在后续介绍 API 的章节再进行细致地说明。

5.5 手把手教你做 Demo——用表单完善通讯录

上一章开发了一款简单的通讯录，也许读者对那个实例不太满意，在重点学习了丰富的小程序表单组件后，用于完善录入功能再合适不过。

将之前的 button 按钮放入 form 组件中，并在 form 组件里放置三个 input 组件，用于接受用户的输入，通过 form 来组合输入组件，可以极大地简化代码量以及开发逻辑。通过本章的学习，读者应该能熟练使用 form 组件来获得其内部输入组件的内容。

按照这个思路，将 index.wmxl 文件的内容改写为：

```
    <view>
        <scroll-view scroll-y style="height:300rpx">
            <view wx:for="{{contactData}}" wx:key="id">
                    <text>{{item.id}}</text>
                    <text>| {{item.name}}</text>
                    <text>| {{item.phone}}</text>
                    <text>| {{thisYear-item.birthday}} </text>
                    |   <icon   data-index="{{index}}"   type="cancel"
size='14' bindtap="deleteMe" />
            </view>
        </scroll-view>
        <!--通 form 表单组件来获得用户的输入-->
        <form bindsubmit="submitFun">
            <input class="ipt" name="name" type="text" placeholder="请
输入用户名" />
            <input class="ipt" name="phone" type="text" placeholder="
请输入手机号" />
            <input class="ipt" name="birthday" type="text" placeholder= "
请输入出生年份" />
            <button class="theBtn"
                form-type="submit"
                >
```

```
            新增
        </button>
    </form>
</view>
```

其中我们稍稍优化了一下样式，在 index.wxss 文件中编写如下代码：

```
.ipt{
    margin:10rpx;
    margin-top: 50rpx;
    padding:10rpx;
    border-bottom:1px solid #CCCCCC;
}

.theBtn{
    margin-top:100rpx;
    height:70rpx;
    font-size:29rpx
}
```

于是，只需在 js 文件中实现 form 组件 bindsubmit 属性绑定的 submitFun 事件，因此在 js 文件中增加 submitFun 方法的实现：

```
/**
     * form 表单的 submit 事件
     */
    submitFun:function(e){
        console.log('提交了',e)
        //由于不能直接操作初始化对象 data 中的值，我们需要创建一个副本进行修改
        var newContactData=this.data.contactData
         //其中的 id 值我们为了唯一，我们为其赋予数组长度+1
        var newItem={
            id:newContactData.length+1,
            name:e.detail.value.name,
            phone:e.detail.value.phone,
            birthday:e.detail.value.birthday
        }
        newContactData.push(newItem)
        this.setData({
            contactData:newContactData
        })
    }
```

保存代码，即可在视图模式下得到如图 5.37 所示的效果。

图 5.37

在该界面输入相应的信息，然后点击“新增”按钮，即可完成将用户输入的内容保存到通讯录列表里。这里的通讯列表依然非常简陋，但是进一步实现了新增用户输入的内容。读者可能会说，这个只是把数据保存在了数组变量里，重新运行程序，就什么都没有了，又有什么意义呢？是的，这里应该能进一步想到，要让这个通讯录具备基本的实用性，需要将数据保存起来，而实现这个功能，则需要借助数据缓存相关的 API，因此在后面章节，将了解相关功能。

5.6 本章要点总结

本章重点介绍了表单组件的使用。

通过熟悉小程序的每个表单组件的使用，也从整体上认识了小程序的默认形态，相信读者脑海中也通过这些组件已经模拟了不少应用场景，做过前端开发的同学想必通过应用 form 组件，也熟悉了如何最大程度上简化表单输入逻辑和代码量，而在本章中，应该重点掌握这一点。

在使用 audio 和 video 组件时，也接触了一些 API 的使用，初步地了解到了 API 的使用方法。

在更多的使用场景中，我们需要更多的使用 API 来实现一些比较复杂的功能。下一章开始学习 API，相信通过 API 的学习，读者能更为深入地掌握微信小程序的开发。

第 6 章

小程序 API（1）：网络、媒体和缓存

API 是获取平台功能的重要途径。微信小程序提供了非常丰富的 API 接口，对应了很多常见的应用场景。和组件的学习类似，小程序 API 的使用也有比较统一的使用规范，了解这些规范对于更快地上手有很大帮助。

6.1 小程序接口规范

小程序接口规范如下。

（1）wx.on 开头的 API 接口通常用来监听事件的发生，例如接口 wx.onSocketError(callback)用来监听 WebSocket 错误，通常可以放在 Page()函数的周期函数 onLoad 或者 onReady 来注册一个方法，统一处理 WebSocket 异常；

（2）除非有特殊约定，否则所有的 API 接口都接受一个 OBJECT 作为参数，OBJECT 参数的内容则根据 API 的功能有所不同；

（3）基本上所有的 API 接口参数 OBJECT 都可以指定 success, fail, complete 三个方法，分别用来处理调用成功、失败、完成三个状态的情况。这三个方法的具体用法官方文档也提供了如表 6.1 所示的说明。

表 6.1　大多数情况 API 接口参数 OBJECT 接受的参数说明表

参数名	类　型	必　填	说　　明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

在小程序里最有特色的回调为 complete 方法，该方法在调用接口后，无论是成功或者失败都会被调用，这非常有利于用户在调试时验证方法是否已经被调用，因为实际上在某些情况下，会有长时间无响应等特殊情况让我们获取不到接口的调用情况，有了 complete 方法，我们可以精确地定位到接口是否被正确调用。

这里需要读者注意的是，我们在公测阶段的开发者工具中使用 API 接口时，碰到过不少次使用接口后确实连 complete 方法也不被调用的情况，并且也不会报错，就像是这个接口调用代码被忽略了一样，并且尝试多次清理缓存，并重新编译也不起作用。目前来看，这种情况是由于开发者工具自身的一些不完善导致，彻底关闭开发者工具，并重新启动项目通常可以解决这个问题。相信开发者工具在后续的更新中会不断地完善这个问题。

6.2　网络

利用小程序网络接口进行请求。

6.2.1　发起请求

对于前端开发者，用 AJAX 技术发起异步网络请求非常常见的需求。从 WEB2.0 时作为热门技术出现，到现在成为 WEB 技术里最为核心的基础功能，jquery 或者 zepto 框架中的$.ajax()是我们熟悉的方法。类似于现在比较热门的 react 和 vue 框架，微信小程序也提供了极为精简的 API 封装：

```
wx.request(OBJECT)
```

按照之前的说明，OBJECT 参数可以指定 success，fail 及 complete 回调方法，另外也接受类似$.ajax()函数常见的参数，详细参数说明如表 6.2（官方文档提供）所示。

表 6.2　接口参数说明列表

	参数名	类　型	必　填	说　　明
1	url	String	是	开发者服务器接口地址
2	data	Object、String	否	请求的参数
3	header	Object	否	设置请求的 header，header 中不能设置 Referer
4	method	String	否	默认为 GET，有效值：OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT
5	success	Function	否	收到开发者服务成功返回的回调函数，res = {data: '开发者服务器返回的内容'}
6	fail	Function	否	接口调用失败的回调函数
7	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

（1）url 是异步请求的地址，在小程序中，url 被要求必须是 https 协议的地址，并且在一个小程序中也只能有最多 5 个网络请求，在 url 中请求的地址域名也必须是在微信小程序开发者后台中指定的 request 合法域名，对于这一点，小程序的限制是比较多的。

（2）data 是请求的参数，写法通常为：

```
data: {
    id: '' ,
    name: ''
  },
```

（3）header 是 http 请求头，我们通常称之为 header，通常用于向服务器指定请求的一些额外参数。小程序规定不能设置 Referer。

（4）method为异步请求的类型，最常见的是POST和GET，而小程序又支持OPTIONS, HEAD, PUT, DELETE, TRACE, CONNECT，如果不指定该参数，则默认为GET请求。

（5）如果做过前端开发，应该对 success 方法非常熟悉，即该请求成功完成后会调用的回调方法。在小程序中，从服务器获得的数据是通过 success 回调方法的参数对象传入的，如果参数对象我们指定为了 res，则可以通过 res.data 来得到服务器返回的数据。

（6）fail 则用来指定请求失败时调用的回调方法，其实 fail 方法也会传入参数

对象，只是该参数对象只有一个 errMsg 属性，并且该属性页的错误信息也非常有限。

（7）complete 方法是小程序 API 接口最有特色的回调方法，无论网络请求是否成功，该方法都会被调用，其传入的参数对象在请求成功和失败时的结构会有所不同，成功时返回的结构与 success 的参数一致，而失败时与 fail 回调的参数一致。

wx.request 方法的通用方法如下：

```
wx.request({
      url: 'http://www.xxx.com',
        method:'GET',
        data: {
          id: '1' ,
          type: '50'
      },
      header: {
          'content-type': 'application/json'
      },
      success: function(res) {
        console.log(res.data)
      },
      fail:function(res){
        console.log(res)
      },
      complete:function(res){
        console.log(res)
      }
    })
```

本例中的 http://www.xxx.com 仅为示例，并非真实接口，读者使用该地址会触发 fail 和 complete 回调方法。另外本例中的 header 和 method 指定的都是默认值，如果删除，其运行效果是一样的，这里也是为了读者便于充分理解该 API 的使用。

网络相关的 API 接口，在实现时需要服务端的配合才能够亲手尝试，这一点会在本章节的手把手环节来通过简单的配置一个本地 node.js 环境进一步熟悉。

6.2.2　上传、下载

在讲解上传之前，先来看一个在 html 中实现上传的代码：

```
    <form     method="post"     action="http://www.xxx.com/server.php"
enctype="multipart/form-data">
    <input name="theFile" type="file" id="f2">
    <input type="submit" name="submit">
    </form>
```

这一段在 html 中实现上传的代码，我们应该非常熟悉，用户在 file 类型的上传表单组件中选择本地文件，并点击 sumbit 按钮后，会将选择的文件提交到 form 表单组件属性 action 指定的 http://www.xxx.com/server.php 地址中去处理，并且该地址中的 server.php 也要相应的实现接收并存储上传文件的功能。

这样的上传功能，在很多需求里也是不可避免的。

小程序则提供了 wx.uploadFile 接口方法来实现上述 html 的上传功能，在学习该接口时，对应着 html 的上传逻辑则能很好地理解其使用方法。该接口的语法为：

```
    wx.uploadFile(OBJECT)
```

其 OBJECT 对象接受如表 6.3 所示的参数（官方文档提供）。

表 6.3　接口参数说明列表

	参　数	类　型	必　填	说　　明
1	url	String	是	开发者服务器 url
2	filePath	String	是	要上传文件资源的路径
3	name	String	是	文件对应的 key，开发者在服务器端通过这个 key 可以获取到文件二进制内容
4	header	Object	否	HTTP 请求 Header, header 中不能设置 Referer
5	formData	Object	否	HTTP 请求中其他额外的 form data
6	success	Function	否	接口调用成功的回调函数
7	fail	Function	否	接口调用失败的回调函数
8	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

如果对 html 的 form 表单实现上传比较熟悉，其和 html 的表单上传对比来看，会更容易理解一些。

（1）url，该 url 对应于 form 表单的 action 属性，指定处理服务端的页面地址。

（2）filePath，需要上传到服务器上的本地文件路径，可以通过 wx.chooseImage 等接口来获取上传的文件路径。

（3）name，和 form 表单中的组件的 name 作用是一样的，即标识这个组件提交的内容，服务端会根据这个名字得到具体 post 的数据。

（4）header，和 wx.request()接口里的一样，是 http 请求头，用于向服务器指定请求的一些额外参数。同样不能设置 Referer。

（5）formData，和 wx.request()接口的 data 的作用一致，可以额外传递一些参数。

剩余的 success，fail 及 complete，和之前接口的作用完全一样，就不再赘述。

使用该接口的代码通常为：

```
wx.chooseImage({
  success: function(res) {
var temp= res.tempFilePaths
//这里则使用了 wx.uploadFile 上传接口
    wx.uploadFile({
      url: 'http://www.xxx.com /upload', //仅用于代码示例
      filePath: tempFilePaths[0],
      name: 'uploadedFile',
      formData:{
        'id': '111'
      },
      success: function(res){
        var data = res.data
        //…上传成功后的操作
},
fail:function(res){
  //…上传失败后的操作
        console.log(res)
      },
complete:function(res){
  //接口调用后的验证
        console.log(res)
      }

    })
```

```
    }
})
```

对于这类接口的使用，建议大家在使用时每次都把 success，fail 及 complete 回调都实现了，即便当时没有时间处理 fail 的情况，也最好实现一个方法，并加入//TODO: 注释标记，并在完成后考虑如何处理该操作失败的流程，这样可以让代码更稳健，也能让代码的逻辑更为清晰。

与 wx.uploadFile(OBJECT)上传 API 对应，小程序提供了 wx.downloadFile (OBJECT) 接口 API 用来实现文件的下载，对于移动端而言，有了这个 API 能够拓展很多应用场景。

对于 wx.downloadFile(OBJECT) 接口 API，其 OBJECT 对象接受如表 6.4 所示的参数（官方文档提供）。

表 6.4　接口参数说明列表

	参　数	类　型	必　填	必　　填
1	url	String	是	下载资源的 url
2	header	Object	否	HTTP 请求 Header
3	success	Function	否	下载成功后以 tempFilePath 的形式传给页面，res = {tempFilePath: '文件的临时路径'}
4	fail	Function	否	接口调用失败的回调函数
5	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相对于 wx.uploadFile 上传接口，wx.downloadFile 接口更易于理解，使用场景也更多一些。

（1）url，就不用多说了，是用来指定需要下载的资源地址。与其他网络相关的 API 一样，正式使用时，其根域名也是需要在后台指定的。

（2）header，也和之前介绍的网络相关的 API 接口的作用一致。

（3）success，调用成功后会回调的方法，主要是这里的参数对象会带有 tempFilePath，如果参数对象使用 res，则可以使用 res.tempFilePath 来获得下载后的临时文件路径。并且需要注意这里获得的临时文件路径在本次小程序运行期间都可以访问，但是小程序关闭后，该临时文件会被小程序删除，下次打开就无法访问，如果需要在下次打开小程序时继续访问，则需要调用 wx.saveFile 接口 API 来保存在本地。该 API 的方法将在本章做详细介绍。

使用该下载 API 的代码通常为：

```
wx.downloadFile({
        url: 'http://www.xxx.com /xxx.jpg', //仅为示例，并非真实的资源
        success: function(res) {
          console.log('读取成功，文件的临时目录为：'+res.tempFilePath)
        },
        fail:function(res){
          //…上传失败后的操作
                console.log(res)
        },
        complete:function(res){
          //接口调用后的验证
                console.log(res)
        }

    })
```

6.2.3 websocket

虽然异步请求符合我们绝大多数的应用场景，但对于服务器通信实时性要求比较高的场景（例如聊天等），则 AJAX 技术的异步请求模式就无法满足需求，在这种场景下，在 HTML5 规范中，通常会使用 WebSocket 接口来实现，而在小程序中，也提供了这个技术的 API 封装，使用起来比较方便。

在一个微信小程序中，只能创建一个 WebSocket 连接，如果存在一个连接的情况下再次创建，则会关闭之前的连接，再重新创建一个 WebSocket 连接。对于这个限制，实际上有利于降低服务器端负荷，通常也是我们在前端开发中都遵守的一个原则。

对于实现 WebSocket 功能，小程序一共提供了 7 个 API 接口。但是其中操作接口仅仅只有三个，其他四个接口都是 wx.on 开始的监听方法，用来监听 WebSocket 事件。因此创建一个 WebSocket 连接在小程序里非常简单。

在使用 WebSocket 与服务器通讯之前，首先需要与服务器建立连接，其 API 接口方法为 wx.connectSocket(OBJECT)。

OBEJECT 对象接受如表 6.5 所示的参数（官方文档提供）。

表 6.5　接口参数说明列表

	参　数	类　型	必　填	说　明
1	url	String	是	开发者服务器接口地址，必须是 wss 协议，且域名必须是后台配置的合法域名
2	data	Object	否	请求的数据
3	header	Object	否	HTTP Header，header 中不能设置 Referer
4	method	String	否	默认是 GET，有效值为：OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT
5	success	Function	否	接口调用成功的回调函数
6	fail	Function	否	接口调用失败的回调函数
7	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

（1）url，是和服务器创建 websocket 连接的地址，必须是 wss 协议，通常使用 websocket 技术时也都采用该协议。这个地址必须是在后台配置的合法域名。

其他参数的作用和用法和之前的网络相关 API 接口一样，这儿不再详细说明。

在代码中，通常这样来使用该接口。

```
wx.connectSocket({
      url: 'ws://www.xxxx.com',
      complete:function(){
          console.log('完成接口调用')
      },
      success:function(){
        console.log('--连接 websocket 成功')
      },
      fail:function(e){
        console.log('--连接 websocket 失败',e)
      }
})
```

用该接口成功创建服务器的连接后，就向服务器发送数据，也可以从服务器接收数据。如果需要监听 WebSocket 连接打开的事件，则可以使用 wx.onSocketOpen 监听方法。通常实现的代码为：

```
wx.onSocketOpen(function(res) {
```

```
    console.log('WebSocket连接已打开！', res)
})
```

其中传入的对象 res 实际上也得到了一些与连接相关的内容，其结构如图 6.1 所示。

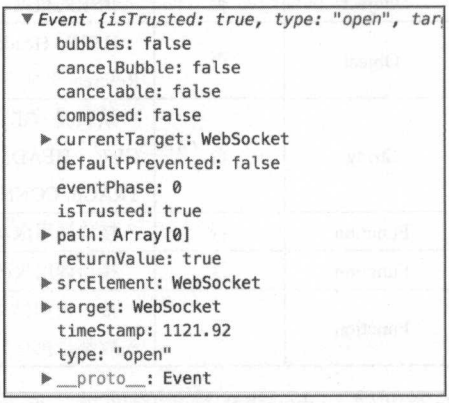


图 6.1

wx.onSocketOpen 接口通常会放在 Page()函数的 onLoad 或者 onReady 周期函数中来使用。

在监听到 WebSocket 连接打开的事件后，就可以继续使用 API 接口 wx.onSocketMessage 监听服务端发来的消息，其传入的事件对象会携带 data 对象，包含服务端发送的消息。并且，比较合理的逻辑应该为：在 wx.onSocketOpen 方法中调用接口 wx.onSocketMessage，因此，通常编写如下代码来监听服务器发来的消息。

```
wx.onSocketOpen(function(res) {
  wx.onSocketMessage(function(res) {
    console.log('收到服务器内容: ' , res.data)
  })
})
```

创建了 WebSocket 连接，并监听到 onSocketOpen 事件后，如果需要向服务端发送消息，则需要使用 API 接口。

```
wx.sendSocketMessage(OBJECT)
```

其 OBJECT 对象接受如表 6.6 所示的参数。

表 6.6　接口参数说明列表

	参　数	类　型	必　填	说　明
1	data	String/ArrayBuffer	是	需要发送的内容
2	success	Function	否	接口调用成功的回调函数
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

下面简要说明一下。

data，为需要发送的内容，可以为字符串 String 或者 ArrayBuffer 类型；其余的参数大家应该比较熟悉，不再详细介绍。

使用该接口向服务器发送消息的代码通常为：

```
wx.sendSocketMessage({
  data: '向服务器发送的字符串内容',
  success:function(){
    console.log('向服务器发送消息成功')
  },
  fail:function(){
    console.log('向服务器发送消息失败')
  },
  complete:function(){
    console.log('调用完成')
  }
})
```

需要注意的是，该接口必须在监听到 onSocketOpen 事件后才能调用，因此通常的做法都是在 onSocketOpen 事件中记录一个标志，用于在调用 sendSocketMessage 接口之前通过该标志判断是否监听到 onSocketOpen 事件。其具体实现，会在后面案例操作时做进一步介绍。

现在已经能够创建 WebSocket 连接，并且向服务器发送消息，对于 WebSocket 技术，还需要能够主动关闭 WebSocket 连接，以节省资源。可以通过如下 API 接口来实现：

```
wx.closeSocket()
```

该接口比较简单，也不接受回调参数，调用该接口方法后，如果存在

WebSocket 连接，则会关闭该 WebSocket 连接，并触发 onSocketClose 事件，如果没有 WebSocket 连接，则不会有任何动作，也不会抛出异常。

无论是主动调用 wx.closeSocket()接口关闭 WebSocket 连接，还是 WebSocket 连接因为其他原因中断，都会触发 onSocketClose 事件，可以通过 wx.onSocketClose 接口来监听该事件，其语法为：

```
wx.onSocketClose(CALLBACK)
```

其用法为：

```
wx.onSocketClose(function(res) {
  console.log('WebSocket 已关闭！')
})
```

这里完成了 WebSocekt 的有关的所有操作，除此之外，小程序又提供了一个用来监听异常的 API：

```
wx.onSocketError(CALLBACK)
```

在代码中的使用方法为：

```
wx.onSocketError(function(res){
  //监听到 WebSocket 异常的处理代码
  console.log('WebSocket 连接打开失败，请检查！')
})
```

6.3 媒体

常见的媒体信息包括图片、视频等消息，下面将在小程序里面的媒体相关接口简要介绍一下。

6.3.1 图片

无论是让用户上传头像或者照片，还是让用户选择照片做处理，或用手机拍摄一张照片，都是一个极为常用的场景。为实现这个场景，小程序提供了 chooseImage 接口，其语法为：

```
wx.chooseImage(OBJECT)
```

其中 OBJECT 对象接受如表 6.7 所示的参数。

表 6.7　接口参数说明列表

	参　数	类　型	必　填	说　　明
1	count	Number	否	最多可以选择的图片张数，默认 9
2	sizeType	StringArray	否	original 原图，compressed 压缩图，默认二者都有
3	sourceType	StringArray	否	album 从相册选图，camera 使用相机，默认二者都有
4	success	Function	是	成功则返回图片的本地文件路径列表 tempFilePaths
5	fail	Function	否	接口调用失败的回调函数
6	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相关参数介绍如下。

（1）count，允许用户最多选择的图片张数，默认为 9，通常在微信中，我们能选择图片的最多张数也是 9，在具体开发中可以根据需求来设置；

（2）sizeType，是一个字符串数组，可以指定' original '原图和' compressed '压缩图，默认两者都有，读者也可以根据需求来指定，例如常见情况下都是指定按照默认的，两者都有，用户在相册里预览图片时，会在左下侧显示“原图”单选框，如图 6.2 所示。

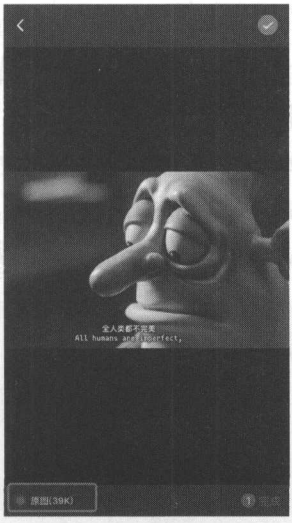


图 6.2

如果只指定' original '或者' compressed '其中一个，则不会显示左下角的原图复选项。

（3）sourceType，也是一个字符串数组，用来指定是' album'从相册选择，还是'camera'从摄像头拍照获取，默认两个都有，如果按照默认值，即两者都有，则会调取如图 6.3 所示的底部菜单。

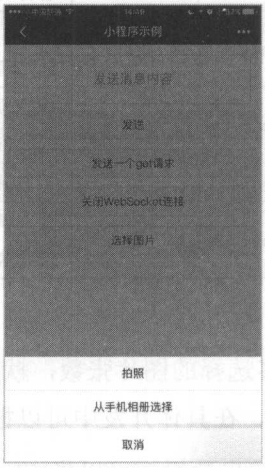


图 6.3

如果只指定' album'，则调用该 API 接口会不显示底部菜单，而直接进入设备相册界面；如果是只指定'camera'，则调用该 API 接口会直接进入设备相机界面。

（4）success，成功后会在其事件参数中传入 tempFilePaths 数组，如果事件对象为 res，则可以通过 res.tempFilePaths 来获得用户选择的图片文件的临时路径数组列表，注意其是一个字符串数组，用户选择多少张图片，会返回多少个数组项。并且获得临时文件路径只在本次应用生存周期内存在，关闭当前小程序会被删除，如果希望下次启动也能够访问，则需要调用接口 wx.saveFile 进行保存。

该 API 接口的使用比较简单，通常的代码会编写为：

```
    wx.chooseImage({
      count: 1, // 默认 9
      sizeType: ['original', 'compressed'], // 可以指定是原图还是压缩图，默认二者都有
      sourceType: ['album','camera'], // 可以指定来源是相册还是相机，默认二者都有
      success: function (res) {
```

```
        // 返回选定照片的本地文件路径列表
        console.log('选择的图片路径为：',tempFilePaths)
    },
    fail:function(res){
      console.log(res)
    },
    complete:function(res){
      console.log(res)
    }
    })
```

用 wx.chooseImage 获取用户选择的图片，就涉及到了显示，虽然可以使用 image 组件在页面上显示，但如果想要有更好地用户体验，还有其他方法吗？对此，小程序还提供了 wx.previewImage 接口 API，封装了非常好的图片预览功能。该 API 的语法为：

```
wx.previewImage(OBJECT)
```

其中 OBJECT 对象接受表 6.8 所示的参数。

表 6.8　接口参数说明列表

	参　数	类　型	必　填	说　明
1	current	String	否	当前显示图片的链接，不填则默认为 urls 的第一张
2	urls	StringArray	是	需要预览的图片链接列表
3	success	Function	否	接口调用成功的回调函数
4	fail	Function	否	接口调用失败的回调函数
5	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相关参数介绍如下。

（1）current，是调用该接口后，首先显示的图片地址，如果不指定，则会显示 urls 参数指定的地址数组中的第一个。

（2）urls，是一个字符串数组，用来指定需要预览的图片地址列表。

通常调用该接口实现图片预览的代码为：

```
previewImageFun:function(){
    wx.previewImage({
      current: '', // 当前显示图片的 http 链接
```

```
        urls: [
          'http://img02.tooopen.com/images/20160603/tooopen_sy_
164101489754.jpg',
          'http://img02.tooopen.com/images/20160603/tooopen_sy_
164101489754.jpg',
          'http://img02.tooopen.com/images/20160603/tooopen_sy_
164101489754.jpg'
        ], // 需要预览的图片 http 链接列表
        success: function(res) {
          console.log('预览图片成功')
        },
        fail:function(res){
          console.log('预览图片失败')
        },
        complete:function(res){
          console.log('预览图片接口调用完成')
        }
      })
    },
```

调用成功后会显示如图 6.4 所示的预览效果。

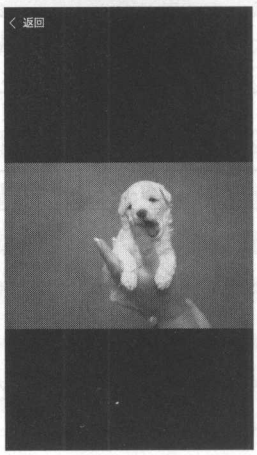


图 6.4

用户可以在该界面通过左划手势来切换指定的多个图片，也可以通过手指进行放大或缩小图片，用户体验非常出色。

要知道如果通过传统的前端技术实现这样的功能还需要不少的代码量，并且

需要解决很多兼容问题，而在小程序中完全消除了这些麻烦 。

除了从用户设备选择图片和预览图片外，小程序还提供了获得图片尺寸的接口 API，其语法为：

```
wx.getImageInfo(OBJECT)
```

OBJECT 接受如表 6.9 所示的参数。

表 6.9　接口参数说明列表

	参　数	类　型	必　填	说　明
1	src	String	是	图片的路径，可以是相对路径，临时文件路径，存储文件路径，网络图片路径
2	success	Function	否	接口调用成功的回调函数
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相关参数介绍如下。

（1）src，是要获取信息的图片路径，可以是相对路径，用 wx.downloadFile 或 wx.chooseImage 等接口得到的临时文件路径，用 wx.saveFile 等接口获得的存储文件路径，甚至是网络图片的路径。

（2）success，其事件对象会传入 width 和 height 属性，可以通过这两个属性得到指定图片的宽度和高度，单位是 px。

用该接口实现获取图片的信息，通常代码可以编写为：

```
wx.getImageInfo({
  src:      'http://img02.tooopen.com/images/20160603/tooopen_sy_
164101489754.jpg',
  success: function (res) {
    console.log('图片的宽度为:'+res.width+',高度为:'+res.height)
  },
  fail:function(res){
    console.log('获取图片信息失败')
  },
  complete:function(res){
    console.log('调用获取图片信息接口完成')
  }
})
```

6.3.2 视频

能够实现让用户从相册选择照片，或者拍摄一张照片的场景后，用户还会有从相册选择视频，或者通过手机摄像头录制视频的需求，对此小程序提供了 API 接口：

```
wx.chooseVideo(OBJECT)
```

其使用思路基本和 wx.chooseImage 类似，对于视频而言有自己的一些区别，OBJECT 接受的参数如表 6.10 所示。

表 6.10 接口参数说明列表

	参 数	类 型	必 填	说明
1	sourceType	StringArray	否	album 从相册选视频，camera 使用相机拍摄，默认为：['album', 'camera']
2	maxDuration	Number	否	拍摄视频最长拍摄时间，单位秒。最长支持 60 秒
3	camera	StringArray	否	前置或者后置摄像头，默认为前后都有，即：['front', 'back']
4	success	Function	否	接口调用成功，返回视频文件的临时文件路径，详见返回参数说明
5	fail	Function	否	接口调用失败的回调函数
6	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相关参数介绍如下。

（1）sourceType，与 wx.chooseImage 接口的 sourceType 参数一样，可以指定是从'album'相册选择，还是从'camera'相机选择，默认两者都有，即['album', 'camera']。

（2）maxDuration，视频拍摄的最大长度，单位为秒，最大长度是 60 秒。

（3）camera，允许的前后置摄像头，默认是两者都有，即['front', 'back']。

（4）success，成功后的回调就不用多说，但是对于本 API 接口，success 的事件对象会传入一些比较重要的参数，具体内容如表 6.11 所示。

表 6.11　接口返回值参数说明列表

	参　数	说　明
1	tempFilePath	选定视频的临时文件路径
2	Duration	选定视频的时间长度
3	size	选定视频的数据量大小
4	height	返回选定视频的长
5	width	返回选定视频的宽

可见其除了得到选取视频的临时目录 tempFilePath 以外，还能得到与视频有关的一些信息。

（1）tempFilePath，从相册选取或者用摄像头录制视频后保存自本地的一个临时目录，该目录同样只能在本次应用周期内有效，如果需要继续有效，则需要通过调用 wx.saveFile 接口来保存。

（2）duration，选定或者录制视频的长度，单位为秒。

（3）size，选定或录制视频的文件大小。

（4）height，选定或录制视频的高度。

（5）width，选定或录制视频的宽度。

通常调用该接口的代码需要编写为：

```
    wx.chooseVideo({
        sourceType: ['album','camera'],
        maxDuration: 60,
        camera: ['front','back'],
        success: function(res) {
            console.log(res)
        },
        fail:function(){
          console.log('获取视频失败')
        }
})
```

运行该段代码后，录制一段小视频，打印出的视频文件信息如图 6.5 所示。

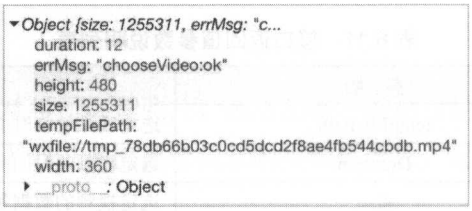


图 6.5

6.3.3 录音

在小程序中实现录音功能也是十分简单的，通过 wx.startRecord 即可非常方便的调用微信提供的录音功能，该接口的语法为：

```
wx.startRecord(OBJECT)
```

其中 OBJECT 对象接受的参数如表 6.12 所示（官方文档提供）。

表 6.12 接口参数说明列表

	参 数	类 型	必 填	说 明
1	success	Function	否	录音成功后调用，返回录音文件的临时文件路径，res = {tempFilePath: '录音文件的临时路径'}
2	fail	Function	否	接口调用失败的回调函数
3	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

一共就三个参数，其中需要学习的就是 success 方法的事件对象，如果对象为 res，则可以通过 res.tempFilePath 获取到录音文件的临时路径。和其他几个接口获取的临时路径一样，该临时路径也会随着小程序关闭而失效，所以如果需要下次启动时继续有效，仍需调用 wx.saveFile 方法来实现。

录音开始后，如果需要主动停止录音，则需要调用另一个 API 接口，语法为：

```
wx.stopRecord()
```

该 API 接口非常简单，不接受任何参数，调用后，如果已开始录音，则会停止。

所以通常录音功能会编写如下代码：

```
wx.startRecord({
  success: function(res) {
```

```
    var tempFilePath = res.tempFilePath
  },
  fail: function(res) {
    //录音失败
  }
})
setTimeout(function() {
  //结束录音
  wx.stopRecord()
}, 10000)
```

录音完成后，还需要一个方法来播放这个语音，不用担心，小程序也封装了对应的 API 接口，下一节就会介绍。

6.3.4　音频播放控制

该接口主要用于播放用 wx.startRecord 接口录制的语音，其语法为：

```
wx.playVoice(OBJECT)
```

其中 OBJECT 对象接受如表 6.13 所示的参数（官方文档提供）。

表 6.13　接口参数说明列表

	参　数	类　型	必　填	说　明
1	filePath	String	是	需要播放的语音文件的文件路径
2	success	Function	否	接口调用成功的回调函数
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

该参数列表中需要了解 filePath 参数，是用 wx.startRecord 接口得到的临时文件路径，或者用 wx.saveFile 方法保存的永久路径。

因此调用该接口的代码通常为：

```
wx.startRecord({
  success: function(res) {
    var tempVoiceFilePath= res.tempFilePath
    wx.playVoice({
      filePath: tempVoiceFilePath,
success: function(){
```

```
        console.log('播放语音成功 ')
    },
    fail:function(){
        console.log('播放语音失败')
    }

    })
  }
})
```

成功播放语音后，如果需要暂停语音，则可以直接使用接口：

```
wx.pauseVoice()
```

该接口也不需要传入任何参数。

暂停语音播放后，如果再次调用 wx.startRecord 接口播放同一个文件，则会从暂停的位置继续播放。

如果需要从头播放，则需要调用如下接口停止播放：

```
wx.stopVoice()
```

该接口也不需要传入任何参数，如果有语音在播放，调用该接口即会停止语音的播放。

通过录音和音频播放控制接口，便可以在小程序中非常方便地实现微信全部的语音功能，结合 Websocket 相关接口，甚至于开发一个类似微信的语音 IM 功能在理论上也是完全可行的，读者可以自行构思一下开发的思路和架构。

6.3.5 音乐播放控制

小程序提供了背景音乐的播放控制功能，该功能是微信自带的音乐播放器，播放后会出现在微信的顶端，如图 6.6 所示。

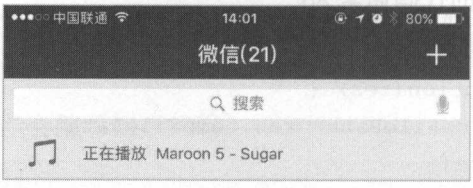


图 6.6

实现该功能则只需调用该接口来进行背景音乐的播放，其语法如下。

```
wx.playBackgroundAudio(OBJECT)
```

其 OBJECT 对象接受如表 6.14 所示的参数（官方文档提供）。

表 6.14　接口参数说明列表

	参　数	类　型	必　填	说　明
1	dataUrl	String	是	音乐链接
2	title	String	否	音乐标题
3	coverImgUrl	String	否	封面 URL
4	success	Function	否	接口调用成功的回调函数
5	fail	Function	否	接口调用失败的回调函数
6	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相关参数说明如下。

（1）dataUrl，需要播放的音乐文件的链接；

（2）title，需要播放的音乐标题，即图 6.6 中所示的播放音乐歌名、歌手等信息；

（3）coverImgUrl，播放音乐的封面图片地址，这个封面会显示在用户点击图 6.6 中所示的播放控制条后显示的播放界面，如图 6.7 所示。

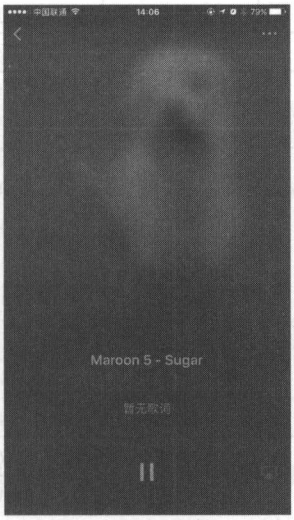


图 6.7

通常实现调用该接口来背景播放的代码为：

```
wx.playBackgroundAudio({
```

```
        dataUrl:
'http://5.1015600.com/2014/ring/000/118/28b0e17cfab0136677648b39cb8b
7fbc.mp3',
        title: 'Maroon 5 - Sugar',
        coverImgUrl:    'http://img02.tooopen.com/images/20160603/
tooopen_sy_164101489754.jpg',
        success:function(){
          console.log('播放成功')
        },
        fail:function(){
          console.log('播放失败')
        }
    })
```

成功播放背景音乐后，就可以通过 API 接口来获得正在背景播放的音乐的状态信息。

```
wx.getBackgroundAudioPlayerState(OBJECT)
```

其 OBJECT 对象接受如表所示的参数如表 6.15 所示。

表 6.15　接口参数说明列表

1	success	Function	否	接口调用成功的回调函数
2	fail	Function	否	接口调用失败的回调函数
3	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success，调用成功的回调函数，状态信息通过 success 函数的事件对象传入，内容如表 6.16 所示。

表 6.16　接口返回参数说明列表

	参　数	说　明
1	duration	选定音频的长度（单位：s），只有在当前有音乐播放时返回
2	currentPosition	选定音频的播放位置（单位：s），只有在当前有音乐播放时返回
3	status	播放状态（2：没有音乐在播放，1：播放中，0：暂停中）
4	downloadPercent	音频的下载进度（整数，80 代表 80%），只有在当前有音乐播放时返回
5	dataUrl	歌曲数据链接，只有在当前有音乐播放时返回

通常调用该接口需要编写如下代码：

```
wx.getBackgroundAudioPlayerState({
      success:function(res){
        console.log('播放成功，播放信息为:',res)
      }
})
```

通过该代码可以在控制台打印出获得的背景音乐状态，如图 6.8 所示。

```
▼Object {dataUrl: "http://5.1015600.com/2014/ring/000/118/28b0e17cfab0136677648b39cb8b7fb
 currentPosition: 34, status: 0, downloadPercent: 94…}
   currentPosition: 34
   dataUrl: "http://5.1015600.com/2014/ring/000/118/28b0e17cfab0136677648b39cb8b7fbc.mp3"
   downloadPercent: 94
   duration: 34
   errMsg: "getMusicPlayerState:ok"
   status: 0
 ▶__proto__: Object
```

图 6.8

对应表 6.16 来看，得到的播放状态内容就非常直观了。

除此之外，还可以通过 API 接口：

```
wx.seekBackgroundAudio(OBJECT)
```

来控制背景音乐播放的进度，通过 OBJECT 对象接受的 position 参数来实现，单位为秒。

通常调用该接口的代码为：

```
  wx.seekBackgroundAudio({
    position:15, //快进到第 15 秒的位置
    success:function(){
      console.log('快进背景音乐成功')
    },
    fail:function(){
      console.log('快进背景音乐失败')
    }
  })
```

通过该代码，会快进到背景音乐的 15 秒位置处并进行播放。

如果需要暂停背景音乐的播放，可以调用 API 接口：

```
wx.pauseBackgroundAudio()
```

不需要传入任何参数。

如果需要继续播放，则需再次调用 API 接口 wx.playBackground Audio (OBJECT)，并在 OBJECT 对象的 dataUrl 参数中传入同样的音乐地址，即可从暂停的位置继续播放。

如果需要停止背景音乐的播放，则可以通过调用 API 接口：

```
wx.stopBackgroundAudio()
```

同样不需要传入任何参数。

调用了该 API 后，再调用 API 接口 wx.playBackgroundAudio(OBJECT)来播放相同地址的音乐，则会重头开始播放。

除了以上控制背景音乐和获得背景音乐状态的接口以外，小程序还提供了监听 API 接口来监听背景音乐的播放状态，分别是：

（1）在背景音乐开始播放时触发的 API 接口：

```
wx.onBackgroundAudioPlay(CALLBACK)
```

通常需要编写的代码为：

```
wx.onBackgroundAudioPlay(function(){
            console.log('背景音乐播放')
})
```

（2）在背景音乐被暂停时触发的 API 接口：

```
wx.onBackgroundAudioPause(CALLBACK)
```

通常需要编写的代码为：

```
wx.onBackgroundAudioPause(function(){
            console.log('背景音乐被暂停')
})
```

（3）以及在背景音乐被停止时触发的 API 接口：

```
wx.onBackgroundAudioStop(function(){
            console.log('背景音乐被停止')
})
```

对于背景音乐的播放，这些 API 接口给我们提供充分挖掘微信背景音乐的功能。

6.3.6　音频组件控制

在前面章节中，已经简单介绍过通过该 API 来控制 audio 组件的播放，实际上该 API 的作用就是用来关联 audio 组件，并通过返回的上下文对象来进一步操作 audio 组件。

主要用法通常是在 onReady 生命周期函数中，使用该 API，通过 audio 组件的 id 属性与该 audio 组件进行绑定，并获得一个上下文对象。代码可以编写为：

```
onReady: function (e) {
    // 使用 wx.createAudioContext 获取 audio 上下文 context
    this.audioContext = wx.createAudioContext('audioId')
  },
```

其中的'audioId'是写在 wxml 文件中 audio 组件的 id 属性，通过这一条语句，this.audioContext 对象就与 id 属性为'audioId'的 audio 组件做了关联，之后就可以通过 this.audioContext 上下文对象操作这个 id 为 audioId 的 audio 组件。

其中这个上下文 context 对象能够做的操作如表 6.17 所示。

表 6.17　context 对象操作方法列表

	方　法	参　数	说　明
1	play	无	播放
2	pause	无	暂停
3	seek	Position	跳转到指定位置，单位 s

直接看一个实现的例子：

```
onReady: function (e) {
    // 使用 wx.createAudioContext 获取 audio 上下文 context
    this.audioContext = wx.createAudioContext('audioId')
  },
//通常绑定在播放按钮上
playAudio:function(){
            this.audioContext.play()
},
//通常绑定在暂停按钮上
pauseAudio:function(){
            this.audioContext.pause()
}
```

```
//通常用来实现一些需要控制进度的特殊需求
seekAudio:function(){
                //快进 audio 组件的播放到第 15 秒的位置开始
                this.audioContext.seek(15)
}
```

6.3.7 视频组件控制

前面章节介绍的 video 组件也介绍了一个 API 接口，即

```
wx.createVideoContext(videoId)
```

其用法和 wx.createAudioContext 是完全一样的，只是通过 wx.createVideo Context 接口获得的上下文对象多了一个发送弹幕的 sendDanmu 方法。

使用方法通常也是在 onReady 生命方法中，使用该 API 与 video 组件的 id 属性进行绑定，并获得一个上下文对象，代码通常编写为：

```
onReady: function (e) {
    // 使用 wx.createVideoContext 获取 audio 上下文 context
    this.videoContext = wx.createVideoContext ('videoId')
  },
```

其中的'videoId'是写在 wxml 文件中 video 组件的 id 属性，通过这一条语句，this.videoContext 对象就与 id 属性为'videoId'的 video 组件做了关联，之后就可以通过 this.videoContext 上下文对象操作这个 id 为'videoId'的 video 组件。

其中这个上下文 context 对象能够做的操作如表 6.18 所示（官方文档提供）。

表 6.18 context 对象支持的操作方法列表

	方 法	参 数	说 明
1	play	无	播放
2	pause	无	暂停
3	seek	position	跳转到指定位置，单位 s
4	sendDanmu	danmu	发送弹幕，danmu 包含两个属性 text, color。

直接看一个实现的例子：

```
onReady: function (e) {
    // 使用 wx.createVideoContext 获取 audio 上下文 context
    this.videoContext = wx.createVideoContext ('videoId')
```

```
  },
  //通常绑定在播放按钮实现视频的播放
  playVideo:function(){
    this.videoContext.play()
  },
  //通常用来实现一些需要控制进度的特殊需求
  pauseVideo:function(){
              this.videoContext.pause()
  },
  //通常
  seekVideo:function(){
              this.videoContext.seek(10)
},
//用来实现发送弹幕
sendDanmu:function(){
            this.videoContext.sendDanmu({
              text: '弹幕文字内容',
      color: '#ffffff '
    })
}
```

6.3.8　文件

在之前介绍 wx.chooseImage 等接口时，接触到了临时路径的概念，并反复提到临时路径会在小程序关闭时失效，如果需要下次启动小程序也能访问，则可以使用本节要介绍的 API 接口：

```
wx.saveFile(OBJECT)
```

其 OBJECT 对象接受如表 6.19 所示的参数。

表 6.19　接口参数说明列表

	参　数	类　型	必　填	说　明
1	tempFilePath	String	是	需要保存的文件的临时路径
2	success	Function	否	返回文件的保存路径，res = {savedFilePath: '文件的保存路径'}
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相关参数说明如下。

（1）tempFilePath，其就是需要保存在本地的临时文件路径，比如通过 wx.chooseImage 得到的临时路径，就可以传入这个参数；

（2）*success 为成功后调用的回调，如果事件对象为 res，那么就可以通过* res.savedFilePath 得到保存到本地的文件路径，并且小程序下次启动后也能访问到。

调用该接口的代码通常为：

```
wx.saveFile({
      tempFilePath: tempFilePath, // tempFilePath 为通过类似 wx.
chooseVideo 接口得到的临时路径
      success: function(res) {
        var savedFilePath = res.savedFilePath // 此时得到的
savedFilePath 路径在小程序下次访问时也能访问
      }
})
```

如果需要获得在本地保存的文件列表，可以调用 API 接口来实现。

```
wx.getSavedFileList(OBJECT)
```

其中 OBJECT 对象接受如表 6.20 所示的参数。

表 6.20　接口返回参数列表

	参　数	类　型	必　填	说　　明
1	success	Function	否	接口调用成功的回调函数，返回结果见 success 返回参数说明
2	fail	Function	否	接口调用失败的回调函数
3	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

该参数列表中，唯一需要关注的是 success 回调方法传入的对象，该对象有两个属性，一个是 errMsg，类型是字符串，记录的是调用的接口的结果；另一个是 fileList 数组，其数组项是一个对象，该对象的属性如表 6.21 所示。

表 6.21　fileList 中数组对象的属性

	键	类　型	说　　明
1	filePath	String	文件的本地路径
2	createTime	Number	文件的保存时的时间戳，从 1970/01/01 08:00:00 到当前时间的秒数
3	size	Number	文件大小，单位 B

相关参数说明如下。

（1）filePath，是文件存储在本地的路径，其实这个路径主要是用来引用这个文件的，供一些 API 调用时传入。

（2）createTime，文件保存的时间戳，是秒数，而不是毫秒。

（3）size，文件的大小，单位是字节，即 bytes。

代码通常编写为：

```
wx.getSavedFileList({
  success: function(res) {
    console.log(res.fileList)
  }
})
```

有了获得本地保存文件列表的接口，并且通过该接口可以了解到这些文件的信息，很自然会想到如果已知一个保存在本地的文件的路径，需要获得关于该文件的信息怎么做呢？可以用另一个接口：

```
wx.getSavedFileInfo(OBJECT)
```

其 OBJECT 对象可以接受的参数如表 6.22 所示。

表 6.22　接口参数说明列表

	参数	类型	必填	说　明
1	filePath	String	是	文件路径
2	success	Function	否	接口调用成功的回调函数，返回结果见 success 返回参数说明
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相关参数说明如下。

（1）filePath，需要获取的文件路径。

（2）success，成功后，可以通过参数对象获得文件的信息，具体如表 6.23 所示。

表 6.23　接口成功后获得的文件信息列表

	参　数	类　型	说　明
1	errMsg	String	接口调用结果
2	size	Number	文件大小，单位 B
3	createTime	Number	文件的保存是的时间戳，从 1970/01/01 08:00:00 到当前时间的秒数

其参数与表 6.21 的一样，通常编写的代码如下：

```
wx.getSavedFileInfo({
  filePath: 'wxfile://somefile', //仅示例用
  success: function(res) {
    console.log(res.size) //文件的大小，单位是字节，bytes
    console.log(res.createTime)，创建文件时间的时间戳
  }
})
```

如果需要删除本地文件，则需要调用 API：

```
wx.removeSavedFile(OBJECT)
```

其 OBJECT 接受如表 6.24 所示参数（官方文档提供）。

表 6.24　接口参数说明列表

	参　数	类　型	必　填	说　明
1	filePath	String	是	需要删除的文件路径
2	success	Function	否	接口调用成功的回调函数
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相关参数说明如下。

filePath，即需要删除的文件路径。

一般情况下，代码需要编写为：

```
wx.removeSavedFile({
      filePath: 'wxfile://somefile', //仅示例用
      complete: function(res) {
        console.log(res)
      }
})
```

小程序还提供了可以在新页面打开文档的 API 接口：

```
wx.openDocument(OBJECT)
```

支持直接打开 doc、xls、ppt、pdf、docx、xlsx、pptx 格式的文档。

该接口的 OBJECT 对象接受如表 6.25 所示的参数。

表 6.25　接口参数说明列表

	参　数	说　明	必　填	说　明
1	filePath	String	是	文件路径，可通过 downFile 获得
2	success	Function	否	接口调用成功的回调函数
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相关参数说明如下。

filePath，注意这个文件路径不能是线上的 URL 地址，如果需要打开线上的 PDF 文件，则需要先通过 wx.downloadFile 下载到本地，再通过得到的临时路径 tempFilePath 来打开，当然也可以通过 wx.saveFile 接口永久保存在本地，再用本 API 接口来打开。

通常调用该 API 接口的代码编写为（官方文档提供示例）：

```
wx.downloadFile({
  url: 'http://example.com/somefile.pdf',
  success: function (res) {
    var filePath = res.tempFilePath
    wx.openDocument({
      filePath: filePath,
      success: function (res) {
        console.log('打开文档成功')
      }
    })
  }
})
```

6.4　数据缓存

在以往的网络世界中，Cookie 无处不在。在前端开发中，使用 Cookie 来存储

个人信息已经非常正常，但随着 Web 技术的突飞猛进，Cookie 的 4kb 容量限制已经无法满足前端开发的需求了。之后随着 HTML5 规范，又为我们提供了 localStorage 接口，现在也逐渐流行了起来。

而在小程序中，提供了 wx.setStorage 接口来实现本地存储，也提供了完整的本地存储方法。小程序允许最大 10MB 的本地存储，这比 Cookie 大太多了，也比 HTML5 规范里的 localStorage 大 1 倍之多，localStorage 通常允许的最大存储限制为 5MB。

另一个独特之处，是每个小程序的数据缓存接口都同时提供了异步和同步方法，使得我们在应用于异步和同步场景时非常方便。

6.4.1 wx.setStorage(OBJECT)

将数据永久存储在本地指定的 key 中，如果指定的 key 已经存在，则会覆盖已存在的值。

该接口的 OBJECT 参数接受如表 6.26 所示的参数。

表 6.26 接口参数说明列表

	参 数	类 型	必 填	说明
1	key	String	是	本地缓存中的指定的 key
2	data	Object/String	是	需要存储的内容
3	success	Function	否	接口调用成功的回调函数
4	fail	Function	否	接口调用失败的回调函数
5	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相关参数说明如下。

（1）key，需要存储在本地的 key 值字符串，注意如果指定的 key 值已经在本地存在，则本地已经存在的 key 值内容会被新的指定的 key 值内容覆盖；

（2）data，需要存储的内容，可以是对象类型，或者是字符串类型。

该接口是一个异步接口，即调用完后，需要在其回调里异步处理完成后的操作，比如存储成功后在 success 回调函数里进行操作。但其实这种模式可能不太适用于常规的后续处理，因此小程序也十分贴心地为用户提供了该方法的同步实现。

通常调用该接口的代码编写为：

```
wx.setStorage({
  key:"key",
  data:"value",
  success:function(){
    console.log('本地存储成功')
  },
  fail:function(){
     console.log('本地存储失败')
   }
})
```

6.4.2　wx.setStorageSync(KEY,DATA)

该方法的功能与 wx.setStorage 接口实现的功能完全一样，只是该接口是同步调用方法，语句执行完成后才会继续执行后续流程，因此也不需要传入回调参数，函数的调用相对更为简洁。

其参数说明如表 6.27 所示。

表 6.27　接口参数说明列表

	参　数	类　型	必　填	说　　明
1	key	String	是	本地缓存中的指定的 key
2	data	Object/String	是	需要存储的内容

其中 key 与 data 参数与 wx.setStorage 接口的用法和作用也完全相同。但是由于没有了成功与失败的回调，作为一个同步接口，需要用 try…catch 结构来捕获可能存在的异常。

因此调用该结构的代码通常编写为：

```
try {
    wx.setStorageSync('storageKey', 'strageValue')
} catch (e) {
    console.log('本地存储失败,失败原因: ',e)
}
```

并且在具体实现中，也强烈建议使用这种 try…catch 结构。

通过上述异步或同步的方法存储本地缓存数据后，可以在调试模式下的 Storage 标签下看到本地缓存数据，如图 6.9 所示。

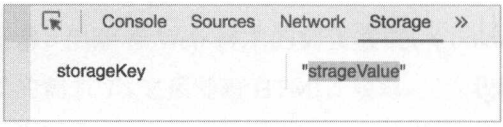


图 6.9

从中可以很直观地看到缓存数据存储情况。

6.4.3 wx.getStorage(OBJECT)

实现了缓存数据的本地存储后，会需要相应的方法去读取，在小程序中使用接口 wx.getStorage(OBJECT)来实现。

其用法和其他接口实现类似，其 OBJECT 对象接受如表 6.28 所示的参数（官方文档提供）。

表 6.28　接口参数说明列表

	参　数	类　型	必　填	说　明
1	key	String	是	本地缓存中的指定的 key
2	success	Function	是	接口调用的回调函数,res = {data: key 对应的内容}
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

相关参数说明如下。

（1）key，指定需要获取的本地缓存数据的 key 名。

（2）success，成功后其回调参数会传入指定 key 名的数据内容，例如参数对象指定为 res，则可以通过 res.data 来得到指定 key 键值的本地缓存数据内容。

调用该接口也非常简单，通常实现的代码为：

```
    wx.getStorage({
      key:'storageKey',
      success:function(res){
        console.log(res.data)
      }
})
```

6.4.4　wx.getStorageSync(KEY)

该方法是获取本地数据缓存的同步方法，其作用和 wx.getStorage(OBJECT) 接口是完全一样的，只是其是同步方法，与异步调用的接口 wx.getStorage 的使用场景有较大区别。

由于该接口为同步接口，因此调用后会直接返回得到本地缓存的数据内容，而无需传入回调方法来得到结果。

其参数 key 的值和 wx.getStorage 传入参数的 key 值用法和作用完全一样，即指定需要获取的本地缓存数据的 key 名。

通常调用该结构的代码为：

```
try {
  var value = wx.getStorageSync('storageKey')
  if (value) {
     // 得到了指定 key 的本地缓存数据内容
  }
} catch (e) {
  // 获取本地缓存时出错，可以通过 e 参数对象来得到出错信息
}
```

6.4.5　wx.getStorageInfo(OBJECT)

该接口用来以异步的方法获得所有已存储的本地缓存内容及大小。

Object 接受三个参数，即 success、fail 和 complete，其用法无需多说，需要介绍的是 success 回调的参数的内容，如表 6.29 所示（官方文档提供）。

表 6.29　接口参数说明列表

	参　数	类　型	说　明
1	keys	String Array	当前 storage 中所有的 key
2	currentSize	Number	当前占用的空间大小，单位 kb
3	limitSize	Number	限制的空间大小，单位 kb

（1）keys，在本地缓存中存储数据的 key 键值名组成的字符串数组。

（2）currentSize，目前所有本地缓存已占用的空间大小，单位是 kb。

（3）limitSize，小程序对缓存的空间限制，通常情况下是 10Mb，用该参数获得的单位是 kb。

通常调用该接口的代码为：

```
wx.getStorageInfo({
  success: function(res) {
    console.log('当前本地缓存中所有的用户数据：',res.keys)
    console.log('当前本地缓存的大小(kb)：',res.currentSize)
    console.log('当前本地缓存的空间限制(kb)',res.limitSize)
  }
})
```

使用该段代码后，在调试环境下可以看到如图 6.10 所示的 log 日志输出。

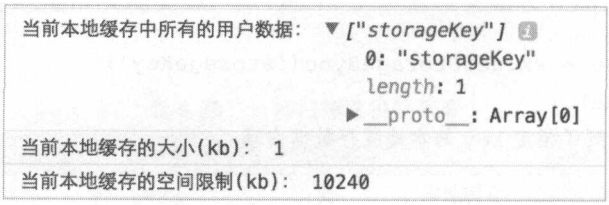


图 6.10

同样，该接口也提供了对应的同步方法。

6.4.6 wx.getStorageSync(KEY)

用来以同步的方法获得所有已存储的本地缓存内容及大小。

无需传入参数，直接用一个变量接受其返回值，即可得到需要的数据，数据结构与用 wx.getStorageInfo 接口得到的一致。

因此代码通常编写为：

```
try {
  var res = wx.getStorageInfoSync()
  console.log('当前本地缓存中所有的用户数据：',res.keys)
  console.log('当前本地缓存的大小(kb)：',res.currentSize)
  console.log('当前本地缓存的空间限制(kb)：',res.limitSize)
} catch (e) {
  console.log('获取当前本地缓存中的数据失败，原因：',e)
}
```

运行该段代码后，在控制台会打印出如图 6.11 所示的日志内容。

图 6.11

可见其效果与通过 wx.getStorageInfo 接口获得的结果是一致的。

6.4.7　wx.removeStorage(OBJECT)

以异步方式从本地缓存删除指定 key 的数据。OBJECT 参数接口如表 6.30 所示参数。

表 6.30　接口参数说明列表

	参　数	类　型	必　填	说　明
1	key	String	是	本地缓存中的指定的 key
2	success	Function	是	接口调用的回调函数
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

主要参数说明如下。

key，指定要删除的本地缓存数据键名。

通常调用本 API 接口的代码编写为：

```
wx.removeStorage({
  key: 'storageKey',
  success: function() {
    console.log('已成功删除指定 key 的本地缓存数据 ')
  }
})
```

运行该代码后，键值名为'storageKey'的本地缓存数据会被删除。

6.4.8 wx.removeStorageSync(KEY)

以同步方式从本地缓存删除指定 key 的数据。

读者一定也已经注意到了，本地数据缓存相关接口里，凡是不带 Sync 后缀的，都是异步调用，实际所有的 API 接口都是异步的，除了带有 Sync 后缀的以外，带有 Sync 后缀的接口是同步接口。并且在本地数据缓存相关接口里，每一个 API 接口都有对应的带 Sync 后缀的以同步方式调用的接口。

其中 key 参数即需要删除的本地缓存的键值名。

调用的代码为：

```
try {
  wx.removeStorageSync('storageKey')
} catch (e) {
  console.log('删除指定的本地缓存数据失败，原因：', e)
}
```

6.4.9 wx.clearStorage()

用来以异步方式删除所有的本地缓存数据。

调用代码为：

```
wx.clearStorage()
```

同理，该接口同样有对应的同步调用方法，即加上 Sync 后缀组成的新接口。

6.4.10 wx.clearStorageSync ()

以同步方式删除所有本地缓存数据。

调用代码为：

```
try {
   wx.clearStorageSync()
} catch(e) {
   console.log('清除本地缓存数据失败，原因：', e)
}
```

异步和同步对于开发者而言最直观的区别就是，异步调用会不等待操作执行

完成，而立即执行后续代码，操作结果由传入的回调函数来处理。而同步调用会在调用的代码处挂起，等待执行完成后，返回结果，再执行后续代码。两种方式各有其优缺点，也有各自适用的场景。关于这两个概念更深入的内容，请读者查阅相关资料。

6.5　手把手教你做 Demo——Websocket 从服务端到小程序

为了更清晰地了解本章介绍的一些重点 API 的使用方法，通过一个实例再熟悉一下。

由于本章介绍的 wx.request 和 WebSocket 相关接口都是与服务端相关的，因此如果我们要熟悉这些 API 必须是有服务端支持才可以，因此，在开始之前先用 node.js 技术实现一个简单的后端服务，其中包括一个 HttpServer 来响应客户端的 get 和 post 请求，以响应 wx.request 发出的网络请求，还包括一个 WebSocketServer，用来响应小程序 WebSocket 相关 API 接口的请求。

6.5.1　安装 Node.js 环境

因此，首先需要安装一个 node.js 环境，这里只介绍直接通过 node.js 官网下载安装的方法，如果读者的电脑已经有了 node.js 环境则可以跳过这一部分，需要了解其他安装方法请自行参阅相关资料。

（1）打开 node.js 官网（网址：https://nodejs.org），在打开的页面中选择右侧的 Current 版本，如图 6.12 所示（此处是以 Mac 为例说明）。

（2）点击进行下载，下载好安装包后，按照默认点击下一步安装至完成即可。

请注意，本例中，终端命令都是以 Mac Os 环境为例进行介绍的，读者如果使用的是其他版本的操作系统，需要留意环境之间的差异。

（3）安装完成后，先在终端运行命令 node –v 以及 npm –v，如果能打印出版本号，说明安装成功，如果出现问题导致安装失败，读者可以尝试再次安装或者使用其他安装方法。

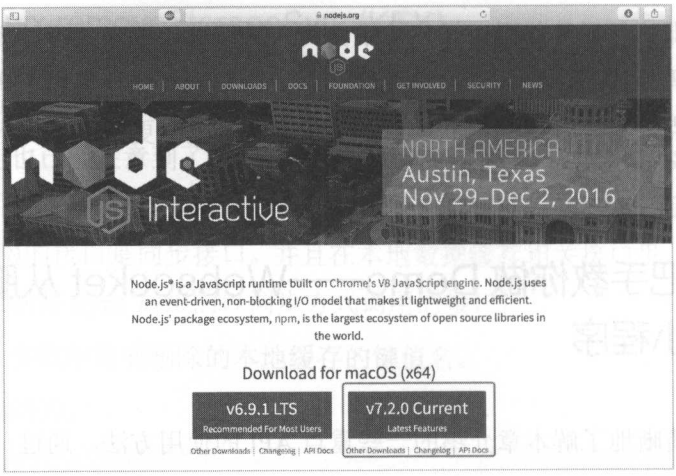


图 6.12

由于国内的 npm 服务不稳定，为了简单起见，我们需要安装一个淘宝的 npm 镜像，官方介绍地址为：

https://npm.taobao.org

安装命令是：

```
npm install -g cnpm --registry=https://registry.npm.taobao.org
```

如果安装遇见权限问题，读者可以尝试加入 sudo 命令：

```
sudo npm install -g cnpm --registry=https://registry.npm.taobao.org
```

或者直接使用命令来修复，

```
sudo chown -R `whoami` /usr/local
```

然后再次尝试运行命令。

也可以直接对 npm 进行换源，具体方法请自行搜索。之后的命令我们都用 cnpm 来讲解，其用法和 npm 是完全一样的，只是带个 c 前缀，如果读者更换了源，去掉 c 即可。

（4）安装好后，先建一个目录，用来存放服务端文件和组件代码。进入新建的这个目录，然后执行 cnpm init，该命令是用来创建一个 node.js 项目，会要求输入项目有关的信息，如项目名称、版本、作者等，可以根据自己情况输入，或者按照默认回车，输入完成后最后会确认一次输入的信息，输入 yes 回车确认，如图 6.13 所示。

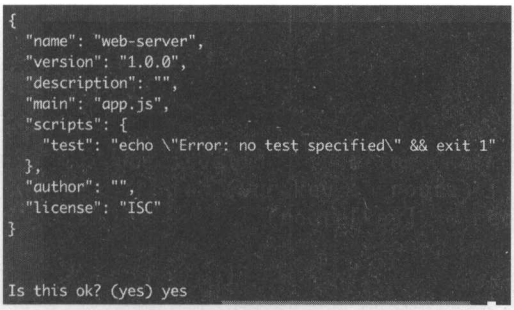


图 6.13

（5）完成后会在该目录创建一个 package.json 文件，这是一个 node.js 项目的描述文件，用来描述项目信息以及重要的包依赖信息。

为了快速实现一个 WebSocket 服务，需要安装一个 node.js 的 websocket 组件。

在当前目录执行命令：

```
cnpm install websocket -save
```

成功后，如图 6.14 所示。

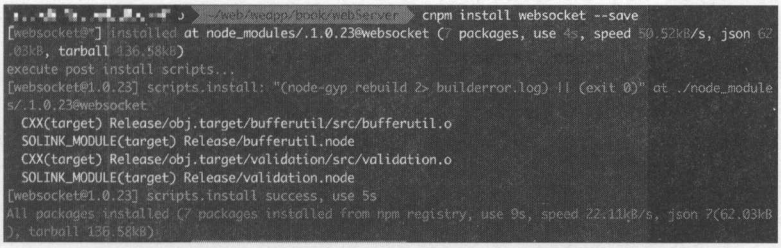


图 6.14

如果读者使用的是 npm 命令而非 cnpm，可能安装成功信息会有所出入，但对使用没有影响。

6.5.2　新建 app.js 文件响应请求

Node.js 安装成功后，即可新建 app.js 文件来进行下一步操作。

（1）安装成功后，新建一个 app.js 文件，并编写如下代码：

```
const WebSocketServer=require('websocket').server;
const http=require('http');
//用来解析 url 地址为 json 类型
```

```
const url = require('url');

/**
 * 一个简单的路由规则
 * 只是用来遍历验证来路的，一般用来提高服务端安全性
 */
var route = {
    '/':'/',
    'test':'/test',
    'api':'/api',
};

/**
 * 对上述来路规则进行验证判断
 * @param  {json 对象}  reqPath [用于验证的服务端来路]
 * @return {Boolean}          [来路是否符合规则]
 */
var isValid = function (reqPath) {
    for (var key in route) {
        if (route[key] == reqPath) {
            return true;
        }
    }
    return false;
};

//创建一个 HTTP Server
let httpServer=http.createServer((request,response) =>{

    var visitedUrl=url.parse(request.url).pathname;
    var methodType =request.method.toUpperCase();
    console.log('request:'+request.url);
    console.log('来路 url:'+visitedUrl);
    console.log('请求类型:'+methodType);
    if(!isValid(visitedUrl)){
        response.writeHead(404, {'Content-Type': 'text/plain;charset=
utf-8'});
        response.end();
    }else{
```

```
        if(methodType==='GET'){
            console.log('这是一个 GET 请求')
            var query = url.parse(request.url, true).query;
            var queryString=JSON.stringify(query);
            console.log('GET 参数:');
            console.log(queryString);
            var responseJson={
                code:200,
                data:{
                    message:'request data success!',
                    date:(new Date())
                }
            }
            response.write(JSON.stringify(responseJson));
            response.end();

        }else if(methodType==='POST'){
            console.log('接收到一个 POST 请求')
            response.write('原来是你, POST');
            response.end();

        }
    }

    console.log('['+(new Date())+'] 访问请求来自 '+request.url);
});

//创建一个 WebSocket Server
var wsServer=new WebSocketServer({
    httpServer:httpServer,
    autoAcceptConnections:true
});

wsServer.on('connect',(connection)=>{
    console.log('socket 连接已成功建立');
    connection.on('message',(message)=>{
        console.log(">> message:",message);
        if(message.type==='utf8'){
            console.log('>> message content from client :'+message.
```

```
utf8Data);
                connection.sendUTF('[from server]'+message.utf8Data);
            }
        })
    });

    httpServer.listen(8010,()=>{
        console.log('['+(new Date())+'] 服务正在监听 8010 端口');
    })
```

这段代码涉及到 node.js 和 es6 的知识，如果读者具备相关知识可以根据注释来理解，如果不具备，也可以直接按本例的步骤来应用，不影响实现效果。

本段代码主要实现两个功能，一个是基本的 http 服务，用来响应 POST 和 GET 请求，另一个是通过创建的 http 服务和 WebSocket 组件，建立一个 WebSocket 服务。

其中比较关键的代码是：

```
    wsServer.on('connect',(connection)=>{
        console.log('socket 连接已成功建立');
        connection.on('message',(message)=>{
            console.log(">> message:",message);
            if(message.type==='utf8'){
                console.log('>>        message        content        from
client :'+message.utf8Data);
                connection.sendUTF('[from server]'+message.utf8Data);
            }
        })
    });
```

这一段代码在 WebSocket 建立好连接后，打印了'socket 连接已成功建立'的字符串，并通过 connection 对象监听了'message'事件，在 message 事件触发时，通过 connection.sendUTF 方法向客户端发送文本消息。

写好代码后，我们需要运行这段代码，在当前目录下运行如下命令：

```
    node app
```

之后如果终端打印出如图 6.15 所示结果，说明代码运行成功。

图 6.15

（2）至此，就建立好了本地的服务端，可以接受 http 的 POST 和 GET 请求，也能处理 WebSocket 连接请求。

6.5.3　编写小程序

接下来开始编写小程序。

这里需要注意的是，如果项目是通过填写 AppID 的方式建立的，那么请求网络接口以及和服务端建立 WebSocket 连接是需要在微信开发者后台配置安全域名，否则会提示如图 6.16 所示的错误。

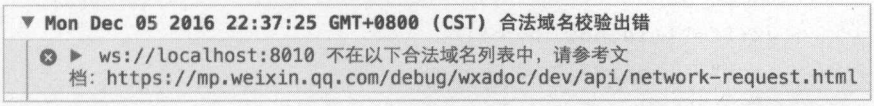


图 6.16

该域名必须是 https 协议的域名，这样就给在本地学习和验证网络相关接口造成了较高的门槛，因此本例最好不要以填写 AppID 的方式来创建微信小程序项目。

即便是目前的代码都是写在带 AppID 的项目中也没有关系，我们只需新建一个“无 AppID”的项目，并将文件目录指定为有 AppID 项目的目录里即可，也就是说，同一个项目可以以带 AppID 和不带 AppID 的形式共存。通过这种方式在本地验证网络功能，在最终上线后，再切回到带 AppID 的版本即可。

（1）创建一个无 AppID 的项目。创建项目的过程就不再赘述，对此不熟悉的读者可以参照前几章的手把手环节。读者可以按照自己的需求创建目录和文件名。

在 wxml 文件中，编写如下代码：

```
<!--WebSocket 消息发送-->
<view class="top">
    <form bindsubmit="submitFun">
        <input class="inputStyle" name="message" placeholder="发送
消息内容"/>
        <button form-type="submit">发送</button>
    </form>

</view>
<!-- 发送一个 get 请求-->
```

```
    <view class="top">
        <button bindtap="requestGetFun">发送一个 get 请求</button>
    </view>
    <!-- 关闭 WebSocket 连接-->
    <view class="top">
        <button bindtap="closeWebSocketFun">关闭 WebSocket 连接</button>
    </view>
    <!-- 选择本地图片并显示-->
    <view class="top">
        <image
            class="imgStyle"
            mode="aspectFit"
            wx:if="{{isShowImage}}"
            src="{{imgSrc}}"></image>
        <button bindtap="choseImageFun">选择图片</button>
    </view>
    <!--选择本地视频并播放-->
    <view class="top">
        <video
            wx:if="{{isShowVideo}}"
            src="{{videoSrc}}"
            controls >
        </video>
        <button bindtap="choseVideoFun">选择视频</button>
    </view>
    <!--实现预览图片功能-->
    <view class="top">
        <button bindtap="previewImageFun">预览图片</button>
    </view>
    <!-- 实现录音功能-->
    <view class="top">
        <button size="mini" bindtap="startRecordFun">录音开始</button>
        <button style="margin-left:20rpx" size="mini" bindtap=
"stopRecordFun">录音结束</button>
    </view>
    <!--实现录音的播放、暂停和停止-->
    <view class="top">
        <button size="mini" bindtap="playRecordVoice">播放录音</button>
        <button style="margin-left:20rpx" size="mini" bindtap=
```

```
"pauseRecordVoice">暂停录音</button>
      <button style="margin-left:20rpx" size="mini" bindtap=
"stopRecordVoice">停止录音</button>
   </view>
```

这段代码布局了比较多的按钮用于实现本章学习的重点功能，通过阅读和理解布局和绑定思路应该已经能够轻易地掌握，加上注释理解更加容易。

（2）虽然布局并没有太顾及美观，但还是稍稍做一些对齐和排列，因此有几个简单的样式需要在 WXSS 文件中编写：

```
.top{
   margin:30rpx 10rpx 10rpx 10rpx;
   text-align: center
}

.inputStyle{
   border:1px solid #CCCCCC;
   border-radius: 7px;
   margin:20rpx;
   padding:20rpx;
   height:100rpx;

}

.imgStyle{
   margin: auto;
}
```

以上都是非常基本的样式内容。

（3）接下来实现关键的逻辑代码，在 JS 文件中编写如下代码：

```
Page({
  isSocketOpened:false,
  url:'ws://localhost:8010',
  tempVoiceFilePath:'',
  data:{
    imgSrc:'',
    isShowImage:false,
    videoSrc:'',
    isShowVideo:false
```

```
  },

  onLoad:function(options){
    // 页面初始化 options 为页面跳转所带来的参数
    var that=this
    this.createSocketConnect()

    //注册监听 websocket 异常
    wx.onSocketError(function(res){
      that.isSocketOpened=false
      console.log('WebSocket 发生异常, 请检查! ')
    })
  },
  /**
   * 通过 wx.connectSocket 接口方法来创建一个 websocket 连接
   * 并通过 wx.onSocketOpen 接口方法监听 socket 连接打开
   */
  createSocketConnect:function(){
    var that=this;
    wx.connectSocket({
      url: this.url,
      complete:function(){
          console.log('完成接口调用')
      },
      success:function(){
        console.log('--连接 websocket 成功')
      },
      fail:function(e){
        console.log('--连接 websocket 失败',e)
      }
    }),
    wx.onSocketOpen(function(res) {
      that.isSocketOpened=true;
      console.log('WebSocket 连接已成功创建! ',res)
      wx.onSocketMessage(function(res) {
        console.log('收到服务器内容: ' , res.data)
      })
wx.onSocketClose(function(){
  that.isSocketOpened=false;
```

```
      console.log('webSocket 连接已关闭')
    })
  })
},

/**
 * 通过 wx.sendSocketMessage 向 websocket 服务端发送消息
 * 发送前先通过变量 isSocketOpened 判断是否已创建了 wocket 连接
 */
sendMessage:function(text){
  if(this.isSocketOpened){
    wx.sendSocketMessage({
      data:text,
      success:function(){
        console.log('向服务器发送消息成功')
      },
      fail:function(){
        console.log('向服务器发送消息失败')
      },
      complete:function(){
        console.log('调用完成')
      }
    })
  }else{
    wx.showModal({
      title: '错误',
      showCancel:false,
      content: '未创建 socket 连接',
      success: function(res) {
        if (res.confirm) {
          console.log('用户点击确定')
        }
      }
    })
  }
},
/**
 * 网络请求
 */
```

```
    requestGetFun:function(){
      wx.request({
        url: 'http://localhost:8010/api?id=1&type=50', //仅为示例，并
非真实的接口地址
          method:'GET',
          data: {
            id: '1',
            type: '50'
        },
        header: {
            'content-type': 'application/json'
        },
        success: function(res) {
          console.log(res.data)
        },
        fail:function(res){
          console.log(res)
        },
      })
    },

    submitFun:function(e){
      this.sendMessage(e.detail.value.message)
    },

    closeWebSocketFun:function(){
      wx.closeSocket()
    },

    /**
     * 调用 wx.chooseImage 接口选择相册或相机图片
     */
    choseImageFun:function(){
      var that=this
      wx.chooseImage({
        count: 1, // 默认 9
        sizeType: ['original', 'compressed'], // 可以指定是原图还是压缩
图，默认二者都有
        sourceType: ['album','camera'], // 可以指定来源是相册还是相机，默
```

```
认二者都有
        success: function (res) {
          // 返回选定照片的本地文件路径列表，tempFilePath 可以作为 img 标签的
src 属性显示图片
          var tempFilePaths = res.tempFilePaths
          that.setData({
            imgSrc:tempFilePaths[0]
          })
          if(tempFilePaths.length>0){
            that.setData({
              isShowImage:true
            })
          }
          console.log('选择的图片路径为: ',tempFilePaths)
        },

      })
    },

    /**
     * 通过调用 wx.chooseVideo 从相册或摄像头获取视频
     */
    choseVideoFun:function(){
      var that = this
      wx.chooseVideo({
        sourceType: ['album','camera'],
        maxDuration: 60,
        camera: ['front','back'],
        success: function(res) {
          console.log(res)
          that.setData({
            videoSrc: res.tempFilePath,
          })
          //判断一下 videoSrc 是否为空是为了确定用户选择了视频
          if(that.data.videoSrc!=''){
            that.setData({
              isShowVideo:true
            })
          }
```

```
        },
        fail:function(){
          console.log('获取视频失败')
        }
      })
    },

    /**
     * 用 wx.previewImage 接口实现图片预览
     */
    previewImageFun:function(){
      wx.previewImage({
        current: '', // 当前显示图片的 http 链接
        urls: [
          'http://img02.tooopen.com/images/20160603/tooopen_sy_
164101489754.jpg',
          'http://img02.tooopen.com/images/20160603/tooopen_sy_
164101489754.jpg',
          'http://img02.tooopen.com/images/20160603/tooopen_sy_
164101489754.jpg'
        ], // 需要预览的图片 http 链接列表
        success: function(res) {
          console.log('预览图片成功')
        },
        fail:function(res){
          console.log('预览图片失败')
        },
        complete:function(res){
          console.log('预览图片接口调用完成')
        }
      })
    },
    /**
     * 调用 wx.startRecord 进行录音
     */
    startRecordFun:function(){
      console.log('录音?')
      var that=this
      wx.startRecord({
```

```
    success: function(res) {
      console.log('完成录音')
      //保存录音的临时文件
      that.tempVoiceFilePath = res.tempFilePath
    },
    fail: function(res) {
      //录音失败
    }
  })

},

/**
 * 播放录音后的文件
 */
playRecordVoice:function(){
  var that=this
  if(that.tempVoiceFilePath!=''){
    wx.playVoice({
      filePath: that.tempVoiceFilePath,
      success: function(){
        console.log('播放语音成功 ')
      },
      fail:function(){
        console.log('播放语音失败')
      }

    })
  }else{
    wx.showToast({
      title: '未录音',
      icon: 'success',
      duration: 2000
    })
  }
},

/**
 * 调用 wx.pauseVoice()接口实现语音播放的暂停
```

```
    */
    pauseRecordVoice:function(){
      wx.pauseVoice()
    },

    /**
     * 调用 wx.stopVoice()接口实现语音播放的终止
     */
    stopRecordVoice:function(){
      wx.stopVoice()
    },

    stopRecordFun:function(){
      wx.stopRecord()
    }
})
```

在这段代码中，封装了一个 createSocketConnect 方法，该方法首先调用 wx.connectSocket 接口与配置的 url 与服务端建立 Websocket 连接，并在 wx.onSocketOpen 事件中监听 wx.onSocketMessage 和 wx.onSocketClose 接口事件。

在之前介绍 API 时讲过，以 on 开头的，通常都是一个事件监听 API，用来监听一个事件的。即该两个方法的功能为在创建一个 Websocket 连接时，监听 Websocket 的消息接收事件和连接关闭事件。

之后在 onLoad 生命周期函数中，调用 createSocketConnect 方法，即可实现与服务端建立 WebSocket 连接的功能，并监听服务端发来的 WebSocket 消息。

运行后控制台会打印出如图 6.17 所示的日志信息，并且在刚才已经开启的 node.js 服务端窗口中打印出如图 6.18 所示的信息。

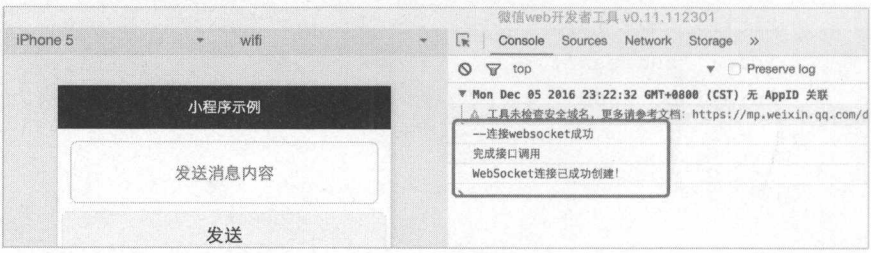


图 6.17

图 6.18

可见已经成功通过小程序与本地服务端建立了 WebSocket 连接。

在代码中的 sendMessage 方法中，首先通过 this.isSocketOpened 变量来判断是否已经建立了 WebSocket 连接，否则如果 WebSocket 连接未建立，发送消息也是毫无意义的，自然会引起异常。

而 this.isSocketOpened 会在 wx.onSocketOpen 事件中设置为 true，在 wx.onSocketClose 事件中设置为 false，以此来标记 WebSocket 的连接状态。

在消息框输入文本内容：hello,websocket，然后点击“发送”按钮，可以得到如图 6.19 所示的效果。

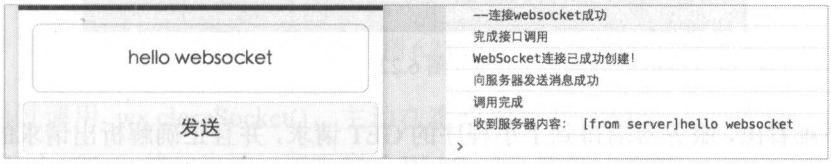


图 6.19

可见通过接口事件 wx.onSocketMessage 收到了服务器发来的消息，而服务端终端窗口则打印出如图 6.20 所示的效果。

图 6.20

至此，我们比较简洁地实现了用小程序与服务端用 WebSocket 进行通信，虽然目前这个服务端是跑在本地的，但是实际上这个服务端完全可以放在远程，并增加一点复杂的机制，来实现更实用的功能，与前面总体思路是一致的，只是实现的逻辑会根据需求复杂一些。

6.5.4　发送 GET 请求

在 requestGetFun 方法中，通过调用 wx.request 接口，实现了对指定 url 地址的 GET 请求，并返回请求结果。

（1）点击“发送一个 GET 请求”按钮，其效果如图 6.21 所示。

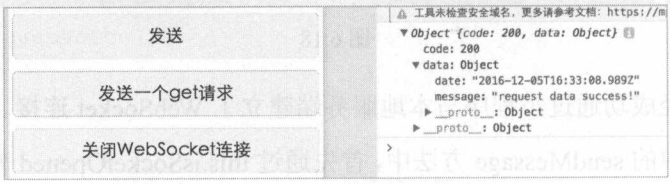


图 6.21

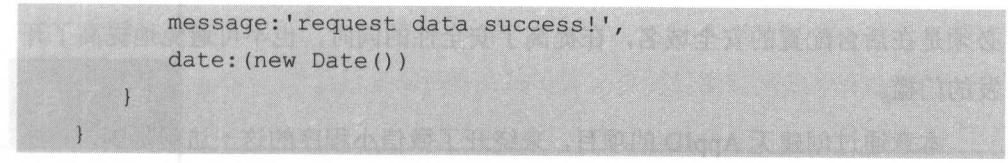
成功请求了 url 地址 http://localhost:8010/api?id=1&type=50，并得到了正确的 json 结果。再看服务终端窗口，打印出如图 6.22 所示的内容。

```
request:/api?id=1&type=50
来路url:/api
请求类型:GET
这是一个GET请求
GET参数:
{"id":"1","type":"50"}
[Tue Dec 06 2016 00:33:08 GMT+0800 (CST)] 访问请求来自 /api?id=1&type=50
```

图 6.22

不难看出，服务器端得到了小程序的 GET 请求，并且正确解析出请求的 URL 地址和参数。实现这段响应 GET 请求的服务端 node.js 代码如下。

```
console.log('这是一个 GET 请求')
var query = url.parse(request.url, true).query;
var queryString=JSON.stringify(query);
console.log('GET 参数:');
console.log(queryString);
var responseJson={
    code:200,
    data:{
        message:'request data success!',
        date:(new Date())
    }
}
response.write(JSON.stringify(responseJson));
response.end();
可以看到我们在小程序中得到的 json 返回值就是这里的
var responseJson={
    code:200,
    data:{
```



```
        message:'request data success!',
        date:(new Date())
    }
}
```

从中我们可以非常清晰的理解 node.js 服务端是如何响应 GET 请求的。

可以将 wx.request 接口调用的参数从 GET 改成 POST，来体验一下其区别。

（2）现在点击“关闭 WebSocket 连接”按钮，控制台打印出了“webSocket 已关闭”的日志信息，如图 6.23 所示。

图 6.23

通过调用 wx.closeSocket()，主动在客户端断开 WebSocket 连接，并通过 wx.onSocketClose 接口方法，监听到了 WebSocket 连接被关闭。

现在通过左侧“编译”按钮重新建立 Websocket 链接，然后在服务终端窗口中通过“control+c”快捷键组合，终止 node.js 进程，然后再看小程序的控制台，也出现了如图 6.22 所示的日志信息，可见 wx.onSocketClose 事件无论是主动断开 WebSocket 连接，还是被动断开，都能准确地监听到事件，因此在 wx.onSocketClose 事件中处理断开的连接，并改变连接状态，是很有必要的。

其他按钮，都是直接调用接口来实现，读者可以结合注释，根据本章相关接口的介绍，通过动手操作，来加深对接口使用方法的理解，并且多动手尝试改动接口参数，来体验调整参数后的效果。

6.6　本章要点总结

本章介绍了网络、媒体、数据相关的 API 接口的使用方法。

其中网络相关接口是小程序开发中最重要的一个环节。但通过本章，我们也了解到对于网络请求，小程序不但限制了只能为 https 请求，还限制了请求的域名

必须是在后台配置的安全域名，在提高了安全性的同时，也不可避免地提高了开发的门槛。

本章通过创建无 AppID 的项目，来绕开了微信小程序的这一访问限制，并通过建立一个 node.js 服务端来实践了网络请求和 WebSocket 相关接口，进一步加深了对接口使用的理解。

通过了解媒体相关的 API，我们也探知到了小程序的强大，以及能够直接使用 API 简化的部分功能，这也是全面了解小程序 API 最大的好处，能够让我们在开发之前，就能想到有哪些 API 提供的功能可以简化我们的开发工作。

数据接口方面，我们接触到了小程序独具特色的同步 API 接口，即以 Sync 结尾的 API，我们能够更轻松地使用同步 API 来适应一些特殊的场景。

本章最后的手把手环节把本章学过的重点 API 接口都梳理了一遍，着重介绍了网络相关 API 的使用方法，读者应该对这部分内容做到熟练掌握。而一部分比较容易理解的 API 没有列入手把手环节的实例中，这部分 API 接口则是留给读者亲自动手尝试。希望读者能花点时间，在本章手把手环节实例的基础上，继续增加应用其他没有应用的一些接口，以加深对接口使用方法和作用的理解。

第 7 章

小程序 API（2）：位置、设备与界面设计

通过上一章的学习，已经熟悉了基本 API 的调用方法，这一章继续讲解常用 API 的使用。

7.1 位置

7.1.1 wx.getLocation(OBJECT) 获取位置

用于获取当前的地理位置和移动速度。如果需要 LBS 应用，则离不开它。

直接看看 OBJECT 对象接受的参数，如表 7.1 所示。

表 7.1 接口参数说明列表

	参 数	类 型	必填	说 明
1	type	String	否	默认为 wgs84 返回 gps 坐标，gcj02 返回可用于 wx.openLocation 的坐标
2	success	Function	是	接口调用成功的回调函数，返回内容详见返回参数说明。
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

type 是获取的地理位置坐标系类型，其默认的 wgs84 坐标系

是国际标准，而 gcj02 则是中国的坐标系标准，不过各坐标系之间可以转换，微信小程序也支持根据指定坐标系来提供具体坐标。

直接通过一段代码来熟悉下该接口。

在 onReady 周期函数中，通过该接口直接获取并打印结果，具体实现代码为：

```
onReady:function(){
    // 页面渲染完成
    wx.getLocation({
        success: function(res) {
            console.log(res)
        },
        fail:function(){
            console.log('接口调用失败')
        },
        complete:function(){
            console.log('接口调用完成')
        }
    })
},
```

运行该段代码，需要注意的是，这段代码在开发者工具调试模式下打印的 res 对象内容与官方提供的并不相符，而必须在设备中运行，并“打开调试”，在“vConsole”中查看调试信息才能看到与官方文档相符的返回结果，如图 7.1 所示。

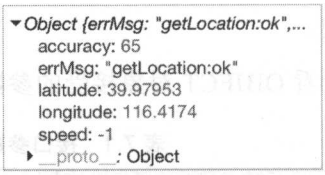


图 7.1　在设备上看到的日志信息

这里可以对照官方提供的表 7.2 来看，就会显得非常直观。

表 7.2　接口返回参数说明列表

	参　数	说　明
1	latitude	纬度，浮点数，范围为-90~90，负数表示南纬
2	longitude	经度，浮点数，范围为-180~180，负数表示西经
3	speed	速度，浮点数，单位 m/s
4	accuracy	位置的精确度

因此，我们获得了自己位置的定位坐标，但是由于没有指定 type 坐标系类型，会默认返回 wgs84 坐标系的坐标，而在小程序中，无论是使用微信内置地图来查看位置，还是使用 map 组件，都需要 gcj02 系的坐标系来展现，因此通常多数情况下会指定 type 坐标系为 gcj02 类型。

7.1.2　wx.chooseLocation(OBJECT) 打开地图选择位置

用来调用微信内置地图，让用户选择位置，在各类以 LBS 服务为基础的应用中会经常碰见，如外卖 APP、电商 APP 等。

官方提供的 OBJECT 对象接受的参数如表 7.3 所示。

表 7.3　接口参数说明列表

	参数	类型	必填	说　明
1	success	Function	是	接口调用成功的回调函数，返回内容详见返回参数说明。
2	cancel	Function	否	用户取消时调用
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

其中比较特殊的是第二个参数 cancel，其在用户点击取消时会被调用。

用一个简单的代码实现下：

```
openChoseLocationFun:function(){
   wx.chooseLocation({
     success:function(res){
       console.log('调用地图选择位置成功, ',res)
     },
     cancel:function(){
       console.log('用户取消选择位置')
     },
     fail:function(){
       console.log('地图选择失败')
     }
   })
 }
```

可以将 openChoseLocationFun 方法绑定在任意界面上的按钮上。

该代码在开发者工具的调试模式下不会有任何效果，必须放在设备上才能

看到，如果需要看到日志信息，还需要打开调试，在设备上运行的效果如图 7.2 所示。

图 7.2

点击“确定”按钮，会打印出如图 7.3 所示的日志信息。

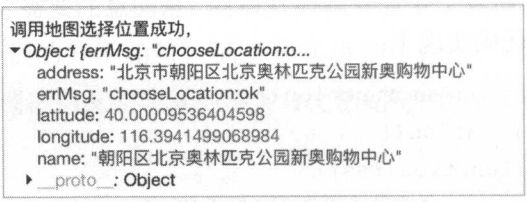


图 7.3

官方文档提供的 success 返回参数说明如表 7.4 所示。

表 7.4　接口返回参数说明列表

	参　数	说　明
1	name	位置名称
2	address	详细地址
3	latitude	纬度，浮点数，范围为-90~90，负数表示南纬
4	longitude	经度，浮点数，范围为-180~180，负数表示西经

如果在地图选择界面选择取消，就会调用 cancel 方法，按照实例代码，会打印出日志消息：'用户取消选择位置'。

7.1.3　wx.openLocation(OBJECT) 使用微信内置地图查看位置

使用微信内置地图查看指定坐标的位置，OBJECT 对象接受如表 7.5 所示参数（官方文档提供）。

表 7.5　接口参数列表

	参　数	类　型	必　填	说　明
1	latitude	Float	是	纬度，范围为-90~90，负数表示南纬
2	longitude	Float	是	经度，范围为-180~180，负数表示西经
3	scale	INT	否	缩放比例，范围 1~28，默认为 28
4	name	String	否	位置名
5	address	String	否	地址的详细说明
6	success	Function	否	接口调用成功的回调函数
7	fail	Function	否	接口调用失败的回调函数
8	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

这个接口的参数比较直观，就不过多介绍，直接代码来了解：

```
openLocationFun:function(){
//先用 wx.getLocation 获得当前位置的经纬度，并且指定是 gcj02 的坐标系
wx.getLocation({
  type: 'gcj02', //返回可以用于 wx.openLocation 的经纬度
  success: function(res) {
    var latitude = res.latitude
    var longitude = res.longitude
    //用 wx.openLocation 接口用内置地图显示指定的坐标
    wx.openLocation({
      latitude: latitude,
      longitude: longitude,
      name:'位置标题', //其实就是位置的标题
      address:'位置的详细描述', //位置标题下面的描述
      scale: 28
    })
  }
  })
```

同样，地理位置相关的接口都需要在设备上运行才能看到效果，运行代码效果如图 7.4 所示。

图 7.4

从图中可以很清楚地了解到 name 和 address 参数的用途，其实就是在底部的显示条里显示这两个文本内容。

7.1.4 wx.createMapContext(mapId) 地图组件控制

该接口用来与界面中的<map/>组件关联，并获得组件的上下文 mapContext 对象，用来对组件进行一些操作，类似接口 wx.createAudioContext(audioId)，并且使用方法和思路也是一样的。

假设在 wxml 文件中，编写了如下的 map 组件：

```
<map id="theMap" show-location />
```

然后可以在 js 文件中通过以下代码获得组件的上下文 mapContext 对象：

```
this.mapContext=wx.createMapContext('theMap')
```

当然，通常，这句代码会写在 onReady 周期函数里。

得到了 mapContext 对象后，就可以实现对<map/>组件的操作，如表 7.6 所示。

表 7.6　mapContext 对象的方法列表

	方　法	参　数	说　明
1	getCenterLocation	OBJECT	获取当前地图中心的经纬度，返回的是 gcj02 坐标系，可以用于 wx.openLocation
2	moveToLocation	无	将地图中心移动到当前定位点，需要配合 map 组件的 show-location 使用

说明如下。

（1）getCenterLocation，用来获取 map 中显示地图当前的中心经纬度，返回的格式是 gcj02 坐标系，可以直接用于接口 wx.openLocation，该接口我们在上一小节已经介绍过，该接口接受一个对象参数，其中的 success 方法会传入 res 参数以返回经纬度。

（2）moveToLocation，将地图的中心移动到当前定位点，即将当前地图恢复到定位位置，注意需要配合 show-location 属性来使用。

getCenterLocation 参数接受一个 object 对象，其支持的参数如表 7.7 所示。

表 7.7　getCenterLocation 方法的 OBJECT 参数列表

	参　数	类　型	必　填	说　明
1	success	Function	否	接口调用成功的回调函数 ，res = { longitude: "经度", latitude: "纬度"}
2	fail	Function	否	接口调用失败的回调函数
3	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success，该方法在回调是会传入 res 参数，我们便可以获得当前地图中心的经纬度。

实现的核心代码如下（在获取了 mapContext 的基础上）。

```
this. mapContext.getCenterLocation({
  success: function(res){
    console.log('当前地图中心的经纬度:',res)
  }
 })
```

运行代码后，可以得到如图 7.5 所示效果。

```
坐标: ▼Object
        latitude: 39.9086611069
        longitude: 116.3972282409668
      ▶ __proto__: Object
```

图 7.5

7.2 设备

获取设备信息可以帮助我们了解用户手机设备的一些状态，根据用户设的状态做对应的处理对于用户体验的提升至关重要。例如基于网络请求的应用里，事先获得用户的联网状态，可以更好地应对无网络的情况，会让用户感觉到体验很友好。

7.2.1 wx.getNetworkType(OBJECT) 获取网络类型

该接口没有特殊参数，只接受常规的 success，fail 和 complete 回调。通过 success 回调函数的对象参数，可以获得用户设备当前的网络类型，如果 success 回调参数指定为 res，则可以通过 res.networkType 获得用户设备的网络类型，其结果为 2g、3g、4g、WiFi 中的一个。

写一个简单的方法来实现：

```
  getNetworkTypeFun:function(){
  /**
   * 通过 wx.getNetworkType 获得用户的网络类型
   */
  wx.getNetworkType({
    success: function(res) {
      console.log('设备当前的网络类型为: ',res.networkType)
    }
  })
}
```

读者可以直接将该方法绑定在按钮或者周期函数中。

通过在设备运行，发现如果将设备飞行模式打开，则控制台会打印出 fail。

7.2.2　wx.getSystemInfo(OBJECT) 获取系统信息

该接口可以获取手机型号、窗口高宽、微信版本号等信息，该接口无特殊参数，只接受常规的 success，fail 和 complete 回调。

具体实现的实例代码为：

```
getDeviceInfo:function(){
  wx.getSystemInfo({
    success: function(res) {
      console.log('用户的设备信息为：',res)
    }
  })
}
```

在开发者工具中运行改代码，则会在调试模式下打印出如图 7.6 所示日志信息。

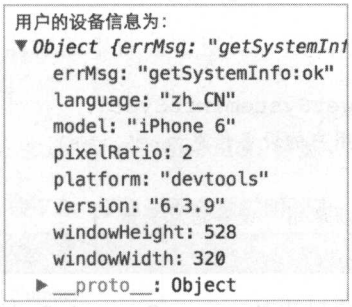


图 7.6

可以对比官方文档提供的 success 回调参数说明如表 7.8 所示。

表 7.8　接口成功返回参数说明列表

	属　性	说　明
1	model	手机型号
2	pixelRatio	设备像素比
3	windowWidth	窗口宽度
4	windowHeight	窗口高度
5	language	微信设置的语言
6	version	微信版本号

根据输出的日志内容来看，实际上比官方文档的说明多出一个 platform 参数。

如果在手机上运行，得到的日志输出如图 7.7 所示。

图 7.7

可见 platform 可以得到用户设备的操作系统平台信息。

7.2.3 wx.getSystemInfoSync () 获取系统信息同步接口

同步接口即以同步的方式来实现获取系统信息。

其代码实现如下：

```
try {
  var res = wx.getSystemInfoSync()
  console.log('用户的设备信息为: ',res)
} catch (e) {
  console.log('获取用户设备信息失败')
}
```

最终打印的日志信息与异步的 wx.getSystemInfo 接口是完全一样的。读者可以亲自尝试运行下代码。

7.2.4 wx.onAccelerometerChange(CALLBACK) 监听重力感应数据

获取设备重力感应数据，可以开发一些比较特殊的功能，如微信“摇一摇”功能就需要获取设备的重力感应数据。

该接口是带有 on 前缀的 API 接口，用来注册事件，因此传入的参数是一个回调方法。回调方法传入的对象如果 res，则我们可以通过 res.x，res.y，和 res.z 来获得 x 轴，y 轴以及 z 轴的数据，回调接口每秒会被调用 5 次。

调用接口的代码为：

```
wx.onAccelerometerChange(function(res) {
```

```
  console.log(res)
})
```

该接口的功能由于涉及到设备芯片，因此也只能在手机上才能得到数据。

运行代码后，在设备上打印的日志消息如图 7.8 所示。

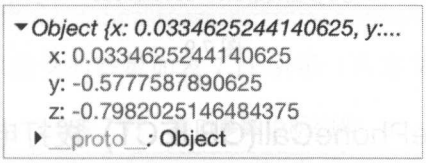


图 7.8

7.2.5　wx.onCompassChange(CALLBACK) 监听罗盘数据

电子罗盘，也叫数字指南针，用来定位北极。

该接口也是一个带 on 前缀的事件监听接口，回调函数会在每秒被调用 5 次，也即每秒获取 5 次罗盘数据。

该事件接口的事件参数会返回一个 direction 参数，如表 7.9 所示。

表 7.9　接口返回参数说明列表

参　数	类　型	说　明
direction	Number	面对的方向度数

调用该接口的代码实现如下：

```
wx.onCompassChange(function (res) {
  console.log(res.direction)
})
```

同样，由于需要特殊的芯片支持，该代码只能在手机设备上才能看到效果。在手机上运行，并将手机指向北边方向后，打印出的日志消息如图 7.9 所示。

可见，返回参数 direction 的值，其实是从北边方向顺时针方向的角度值。

通过该接口实现功能，需要手机芯片的支持，市面上有一些安卓设备是不具备电子罗盘芯片的。

图 7.9

7.2.6 wx.makePhoneCall(OBJECT) 拨打电话

调用该接口，会直接调用用户手机的拨号功能对指定的号码进行呼叫。

其 OBJECT 对象接受如表 7.10 所示的参数（官方文档提供）。

表 7.10 接口参数说明列表

	参 数	类 型	必 填	说 明
1	phoneNumber	String	是	需要拨打的电话号码
2	success	Function	否	接口调用成功的回调
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

phoneNumber，即需要拨打的电话号码，该接口的调用代码则为：

```
wx.makePhoneCall({
  phoneNumber: '10010', //联通客服号码
  success:function(){
    console.log('拨打电话成功')
  },
  fail:function(){
    console.log('拨打电话失败')
  }
})
```

运行后，便会调用手机的拨号功能直接拨打指定的'10010'号码。

7.3 界面

在产品交互体验越来越重要的今天，界面的重要性已无须多说，本章介绍的组件将有助于我们实现一些交互的功能。

7.3.1　交互反馈

完成操作提示用户已完成，或者执行某项操作时，提示用户等待，可以根据场景需要使用接口 wx.showToast 来实现，其具体语法为：

```
wx.showToast(OBJECT)
```

其中 OBEJCT 对象接受的参数如表 7.11 所示（官方文档提供）。

表 7.11　接口参数说明列表

	参　数	类　型	必填	说　明
1	title	String	是	提示的内容
2	icon	String	否	图标，只支持"success"、"loading"
3	duration	Number	否	提示的延迟时间，单位毫秒，默认：1500，最大为 10000
4	success	Function	否	接口调用成功的回调函数
5	fail	Function	否	接口调用失败的回调函数
6	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

（1）title，即提示的文字内容字符串。

（2）icon，提示内容中间会显示一个 icon，其有两种类型，即 success 和 loading。

（3）duration，提示显示的最长时间，单位是毫秒，如果不传，默认为 1500 毫秒，最大为 10000 毫秒，也就是提示最多只能显示 10 秒。

调用该接口的代码通常为：

```
showToastFun:function(){
 wx.showToast({
   title: '操作执行成功',
   icon: success,
   duration: 2000
 })
}
```

运行改代码的效果如图 7.10 所示。

图 7.10

如果将 icon 参数改为 loading 后的效果如图 7.11 所示。

图 7.11

显示了 Toast 提示后，如果需要在其超时时间内提前关闭，可以通过调用 wx.hideToast 接口实现，其语法为：

```
wx.hideToast()
```

不需要传入参数，通常使用的场景是，用 wx.showToast 提示用户等待，并且 icon 类型为 loading 时，我们在等待时间内提前得到了执行的回调，则可以在这个回调中调用 wx.hideToast()接口来结束正在提示等待的 Toast 提示。可以将之前的代码修改为：

```
showToastFun2:function(){
  wx.showToast({
    title: '操作执行成功',
    icon: 'loading',
    duration: 6000
  })
  setTimeout(function(){
    wx.hideToast()
  },2000)
}
```

代码中虽然设定了 6000 毫秒的提示持续时间，但我们通过 setTimeout 在两秒后就提前关闭了 Toast 提示。

Toast 提示也属于模态窗口，即显示了 Toast 提示后，用户是不能进行其他操作的。

如果需要弹出一个对话框，并让用户进行选择，在小程序中可以使用接口 wx.showModal，其语法为：

```
wx.showModal(OBJECT)
```

其 OBJECT 对象接受如表 7.12 所示参数。

表 7.12　接口参数说明列表

1	title	String	是	提示的标题
2	content	String	是	提示的内容
3	showCancel	Boolean	否	是否显示取消按钮，默认为 true
4	cancelText	String	否	取消按钮的文字，默认为“取消”，最多 4 个字符
5	cancelColor	HexColor	否	取消按钮的文字颜色，默认为“#000000”
6	confirmText	String	否	确定按钮的文字，默认为“确定”，最多 4 个字符
7	confirmColor	HexColor	否	确定按钮的文字颜色，默认为“#3CC51F”
8	success	Function	否	接口调用成功的回调函数，返回 res.confirm 为 true 时，表示用户点击确定按钮
9	fail	Function	否	接口调用失败的回调函数
10	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

（1）title，模式对话弹窗的标题字符串；

（2）content，模式对话弹窗的提示文字内容；

（3）showCancel，是否显示取消按钮，默认是 true；

（4）cancelText，取消按钮的文字，默认值是“取消”，最多 4 个字符；

（5）cancelColor，取消按钮文字的颜色，使用 HexColor 格式的十六进制的颜色码，默认为“#000000”，即黑色；

（6）confirmText，确定按钮的文字，默认值是“确定”，最多也是 4 个字符；

（7）confirmColor，确定按钮的文字颜色，使用 HexColor 格式的十六进制的颜色码，默认为“#3CC51F”，即绿色；

（8）success，接口调用成功调用的回调函数，其回调函数的参数对象如果为 res，res.confirm 为 true，则说明用户点击了确定按钮。

先用通过码实现一个模式弹窗对话框，再看参数的使用会更明显一些：

```
showModalFun:function(){
  wx.showModal({
    title: '对话框',
    content: '这是一个模态弹出对话框',
    showCancel:true,
    cancelText:'取消按钮',
    cancelColor:'#99CC33',
    confirmText:'确定按钮',
```

```
      confirmColor:'#CC3300',
      success: function(res) {
        if (res.confirm) {
          console.log('用户点击了确定按钮')
        }
      }
    })
  }
```

在这段代码里，我们应用了所有的参数。绑定该方法到某一个按钮上进行绑定并点击后，会弹出如图 7.12 所示对话框。

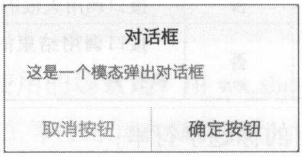


图 7.12

如果把 showCancel 参数设置为 false，则会显示如图 7.13 所示的弹出对话框。

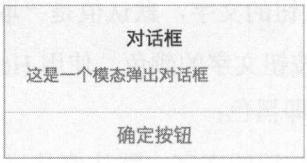


图 7.13

点击“确定按钮”后，会打印出日志消息“用户点击了确定按钮”。

到目前为止，我们学习了没有按钮的模式弹窗接口 wx.showToast，以及带一个或两个按钮的模式弹窗组件 wx.showModal，仔细想一下，有些情况下两个按钮也是不够的，那该怎么办呢？别担心，小程序也为用户提供了任意选择项的显示操作菜单 API 组件。

```
wx.showActionSheet(OBJECT)
```

其 OBJECT 对象接受如表 7.13 所示参数（官方文档提供）。

表 7.13　接口参数说明列表

	参　数	类　型	必　填	说　明
1	itemList	String Array	是	按钮的文字数组，数组长度最大为 6 个
2	itemColor	HexColor	否	按钮的文字颜色，默认为"#000000"

续表

	参　数	类　型	必　填	说　明
3	success	Function	否	接口调用成功的回调函数，详见返回参数说明
4	fail	Function	否	接口调用失败的回调函数
5	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

（1）itemList，菜单项的字符串数组，最多 6 个。

（2）itemColor，菜单项的文字颜色，默认为“#000000”。

（3）success，接口调用成功的回调，其回调会传入如表 7.14 所示的参数。

表 7.14　接口成功返回参数说明列表

	参　数	类　型	说　明
1	cancel	Boolean	用户是否取消选择
2	tapIndex	Number	用户点击的按钮，从上到下的顺序，从 0 开始

（1）cancel，用户是否选择了取消。

（2）tapIndex，用户选择的菜单项下标索引，从 0 开始。

其实现的代码：

```
showActionSheetFun:function(){
    wx.showActionSheet({
    itemList: ['跳过', '继续', '重试'],
    success: function(res) {
      if (!res.cancel) {
        console.log(res.tapIndex)
      }
    }
  })
```

绑定到任意按钮上并点击，显示如图 7.14 所示的操作菜单。

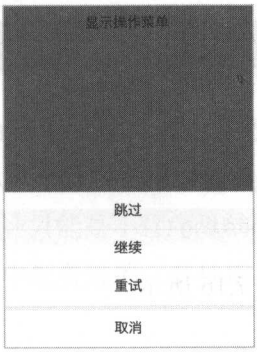


图 7.14

7.3.2 设置导航条

如果需要在运行过程中动态的改变当前页面的标题，或者在标题显示加载动画，则可以使用本节介绍的三个 API 接口。

设置当前页面标题的接口语法为：

```
wx.setNavigationBarTitle(OBJECT)
```

其 OBJECT 对象接受的参数如表 7.15 所示（官方文档提供）。

表 7.15 接口参数说明列表

	参 数	类 型	必 填	说 明
1	title	String	是	页面标题
2	success	Function	否	接口调用成功的回调函数
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

title，要动态设置的标题字符串。

实现的代码非常简单：

```
changeTitleFun:function(){
  wx.setNavigationBarTitle({
    title: '重新设置的标题内容'
  })
},
```

绑定到任意按钮上，并点击按钮后，当前页面的标题则会被改为指定的内容，如图 7.15 所示。

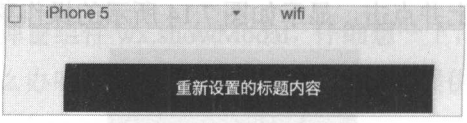


图 7.15

如果要在标题栏显示加载动画，则可以直接调用接口：

```
wx.showNavigationBarLoading()
```

调用后标题栏的效果如图 7.16 所示。

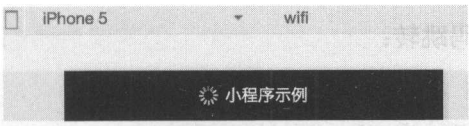


图 7.16

如果需要隐藏加载动画，则可以直接调用接口：

```
wx.hideNavigationBarLoading()
```

通常在有网络请求时，可以在标题栏显示一个加载动画，正如微信一样，会让用户体验更好一些。

7.3.3　导航

小程序的导航就是指在小程序不同页面之间的跳转，在小程序中是不允许跳转到 url 外网地址的，只能在小程序内部页面中跳转。

最常用的保留当前页面跳转的 API 接口为：

```
wx.navigateTo(OBJECT)
```

其 OBJECT 对象接受的参数如表 7.16 所示（官网提供）。

表 7.16　接口参数说明列表

	参　数	类　型	必　填	说　　明
1	url	String	是	需要跳转的应用内页面的路径 ，路径后可以带参数。参数与路径之间使用?分隔，参数键与参数值用=相连，不同参数用&分隔；如 'path?key=value&key2=value2'
2	success	Function	否	接口调用成功的回调函数
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

url，需要跳转的页面地址，并且该 url 地址是可以带参数的，如?id= 1&page=3，这些参数会通过 onLoad 生命周期函数方法的参数传入，如果这个参数对象是 option，则可以在 onLoad 生命周期函数内通过 option 取到当前页面传入的参数。另外需要注意的是，这里的地址只能是小程序里的页面地址，并且是在小程序根目录的 app.json 文件 pages 项中配置过的，其他地址的跳转不会被执行，且要改地址为相对路径。

如果使用如下代码跳转：

```
wx.navigateTo({
  url: '../api-6/api-6?id=1&page=3'
})
```

则在../api-6/api-6 目录下的 api-6.js 文件中需要用如下方法获得地址参数：

```
onLoad:function(options){
  // 页面初始化 options 为页面跳转所带来的参数
  console.log('传入的参数内容',options)
}
```

如果执行上述流程内容，则会得到日志信息如图 7.17 所示。

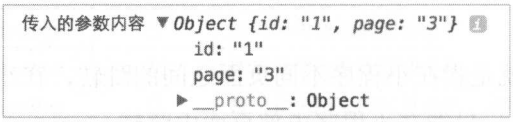


图 7.17

可见其 options 会以 json 格式直接获取到地址参数内容。

wx.navigateTo 接口的主要特点是，跳转去的页面左上方会显示一个返回按钮，用户可以随时返回跳转之前的页面。

跳转后的页面标题如图 7.18 所示。

图 7.18

用该接口实现应用内的页面跳转后，如果需要通过代码控制返回上一级或多级页面，则可以通过接口 wx.navigateBack 来实现，其语法为：

```
wx.navigateBack(OBJECT)
```

其 OBJECT 接受的参数如表 7.17 所示。

表 7.17　接口参数说明列表

	参　数	类　型	默认值	说　明
1	delta	Number	1	返回的页面数，如果 delta 大于现有页面数，则返回到首页。

delta，是返回的页面层级，大于现有的页面层级数，会返回首页。也可以通过 getCurrentPages()来获取当前页面栈，来决定返回几级。

调用接口的例子：

```
wx.navigateBack({
    delta:1
})
```

而如果需要跳转到一个页面后，用户无法再跳转回去，也不在左上方显示返回按钮，则需要借助 API 接口：

```
wx.redirectTo(OBJECT)
```

其 OBJECT 对象接受如表 7.18 所示的参数（官方文档提供）。

表 7.18　接口参数说明列表

	参　数	类　型	默认值	说　明
1	url	String	是	需要跳转的应用内页面的路径
2	success	Function	否	接口调用成功的回调函数
3	fail	Function	否	接口调用失败的回调函数
4	complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

url，是需要跳转的应用内页面路径，并且跳转后用户无法回到跳转之前的页面。和 wx.navigateTo 接口一样，该地址一样可以带地址参数，如?id=1&page=3，也是通过 onLoad 周期函数回调来传入的。

实例代码为：

```
wx.redirectTo({
  url: '../api-6/api-6?id=1&page=3'
})
```

跳转后的页面标题如图 7.19 所示。

图 7.19

从图中可见，页面标题左侧没有返回按钮，用户无法返回先前的页面。

7.3.4　动画

小程序提供了一个特殊的动画接口，用来创建一个动画对象，并通过对象封装的方法调用来创建动画，最后要真正实现还得把这个对象绑定到组件的

animation 属性。虽然看起来接口方法很多，但其实用法都一样，只需根据自己的需要选择调用即可。

该接口的语法为：

```
wx.createAnimation(OBJECT)
```

其 OBJECT 对象如表 7.19 所示的参数（官方文档提供）。

表 7.19　接口参数说明列表

	参　数	类型	必填	说　明
1	duration	Integer	否	动画持续时间，单位 ms，默认值 400
2	timingFunction	String	否	定义动画的效果，默认值“linear”，有效值：“linear”，“ease”，“ease-in”，“ease-in-out”，“ease-out”，“step-start”，“step-end"
3	delay	Integer	否	动画延迟时间，单位 ms，默认值 0
4	transformOrigin	String	否	设置 transform-origin，默认为“50% 50% 0”

（1）duration，动画持续的时间，默认是 400 毫秒。

（2）timingFunction，准确地说应该是过渡效果的速度曲线，与 css3 的 transition-timing-function 属性的功能是一样的，但是注意也是有一些区别的，其中 step-start 与 step-end 是小程序比较特殊的属性，并且小程序也不支持 css3 的 cubic-bezier 自定义。

（3）delay，动画的延迟时间，单位是毫秒，默认值 0。

（4）transformOrigin，是定义动画的原点，与 css3 的 transform-origin 用法与作用都是一致的。

这里需要留意的是，wx.createAnimation 接口的这些参数全部都不是必填，因此完全可以像下面这样创建动画对象：

```
var animation = wx.createAnimation()
```

得到了 animation 动画对象后，就可以使用对象的方法来绘制动画了，这都是变形动画，虽然方法众多，但用法其实都是类似的，学会其中的几项，其他用法都是一样，这里为了便于读者查阅，先把官方提供的列表都列出。函数调用会返回对象本身，因此可以支持链式调用。

animation 对象的样式变换的方法如表 7.20 所示。

表 7.20　animation 对象的样式变换的方法表

	方　法	参　数	说　明
1	opacity	value	透明度，参数范围 0~1
2	backgroundColor	color	颜色值
3	width	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
4	height	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
5	top	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
6	left	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
7	bottom	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
8	right	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值

animation 对象的旋转变换方法如表 7.21 所示。

表 7.21　animation 对象的旋转变换方法表

	方　法	参　数	说　明
1	rotate	deg	deg 的范围-180~180，从原点顺时针旋转一个 deg 角度
2	rotateX	deg	deg 的范围-180~180，在 *X* 轴旋转一个 deg 角度
3	rotateY	deg	deg 的范围-180~180，在 *Y* 轴旋转一个 deg 角度
4	rotateZ	deg	deg 的范围-180~180，在 *Z* 轴旋转一个 deg 角度
5	rotate3d	(x,y,z,deg)	同 transform-function rotate3d

animation 对象的缩放变换方法如表 7.22 所示。

表 7.22　animation 对象的缩放变换方法表

	方　法	参　数	说　明
1	scale	sx,[sy]	一个参数时，表示在 *X* 轴、*Y* 轴同时缩放 sx 倍数；两个参数时表示在 *X* 轴缩放 sx 倍数，在 *Y* 轴缩放 sy 倍数
2	scaleX	sx	在 *X* 轴缩放 sx 倍数
3	scaleY	sy	在 *Y* 轴缩放 sy 倍数
4	scaleZ	sz	在 *Z* 轴缩放 sy 倍数
5	scale3d	(sx,sy,sz)	在 *X* 轴缩放 sx 倍数，在 *Y* 轴缩放 sy 倍数，在 *Z* 轴缩放 sz 倍数

animation 对象的偏移变换方法如表 7.23 所示。

表 7.23　animation 对象的偏移变换方法表

	方　法	参　数	说　明
1	translate	tx,[ty]	一个参数时，表示在 X 轴偏移 tx，单位 px；两个参数时，表示在 X 轴偏移 tx，在 Y 轴偏移 ty，单位 px
2	translateX	tx	在 X 轴偏移 tx，单位 px
3	translateY	ty	在 Y 轴偏移 tx，单位 px
4	translateZ	tz	在 Z 轴偏移 tx，单位 px
5	translate3d	(tx,ty,tz)	在 X 轴偏移 tx，在 Y 轴偏移 ty，在 Z 轴偏移 tz，单位 px

animation 对象的倾斜变换的方法如表 7.24 所示。

表 7.24　animation 对象的倾斜变换的方法表

	方　法	参　数	说　明
1	skew	ax,[ay]	参数范围-180~180；一个参数时，Y 轴坐标不变，X 轴坐标延顺时针倾斜 ax 度；两个参数时，分别在 X 轴倾斜 ax 度，在 Y 轴倾斜 ay 度
2	skewX	ax	参数范围-180~180；Y 轴坐标不变，X 轴坐标延顺时针倾斜 ax 度
3	skewY	ay	参数范围-180~180；X 轴坐标不变，Y 轴坐标延顺时针倾斜 ay 度

animation 对象的矩阵变形的方法如表 7.25 所示。

表 7.25　animation 对象的矩阵变形的方法表

	方　法	参　数	说　明
1	matrix	(a,b,c,d,tx,ty)	同 transform-function matrix
2	matrix3d		同 transform-function matrix3d

读者不需要因为这些表的内容繁多而眼花，下面来逐步介绍，了解后只需根据自己所需使用即可。

既然是动画效果，就必定要与视图层关联，需要让视图层的组件动起来，因此就必定需要一个属性能够和用 wx.createAnimation 接口创建的对象通过一种方法绑定起来，在小程序中，这个属性就是 animation，所有有视图的组件都有这个属性，因此我们需要让一个组件有动效，第一步就是给这个组件增加一个 animation 属性，并和一个变量做绑定，我们这里就和一个叫 animationData 的变量绑定起来，因此，需要在 wxml 中编写如下代码：

```
<view class="aniDom" animation="{{animationData}}">
</view>
```

在一个交互操作后执行动画效果，便于我们清晰地了解创建动画效果的步骤，因此需要再创建一个 button 组件，在 wxml 文件中增加代码：

```
<button bindtap="biggerFun">
    放大
</button>
```

为了让这个空的 view 组件直观可见，在 wxss 文件中编写如下样式代码：

```
.aniDom{
    margin:200rpx auto;
    background-color:  cadetblue;
    height:100rpx;
    width:300rpx;
}
```

完成如下代码后，小程序视图页面的效果如图 7.20 所示。

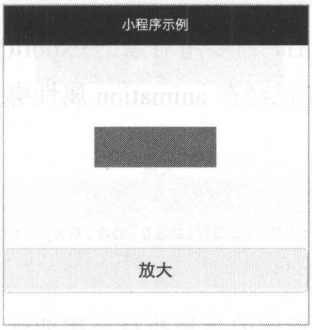


图 7.20

首先，需要在 js 文件中定义好绑定的变量 animationData。

读者对此也已经很熟练了，代码如下：

```
data:{
    animationData:{}
  },
```

接着，需要在初始化时创建一个全局的动画对象。初始化的工作，通常放在周期函数 onShow 里做比较好，并且可以用 this 来创建一个全局的对象，以供其他方法使用。代码如下：

```
onShow:function(){
    // 页面显示
    this.animation = wx.createAnimation({
      duration: 500,
      timingFunction: 'ease-in-out',
```

```
    })
  },
```

这里我们给 wx.createAnimation 传入了两个参数，duration 指定动画时长为 500 毫秒，timingFunction 指定动画效果为：'ease-in-out'。

有了 this.animation 对象，我们就可以调用其相应的动画方法来实现动画了，选择一个比较常见的放大效果 scale，其参数在一个参数时表示 x 轴和 y 轴都同时放大，这就足够我们使用了，这里需要注意的是，调用完任何动画方法之后，都必须调用一个 step()方法来代表动画结束。

于是放大动画就可以编写为：

```
this.animation.scale(2).step()
```

给对象指定完动画效果后，需要用对象的 export()方法来导出动画，并与组件的 animation 属性绑定，我们已经在 animation 属性绑定变量 animationData，因此实现动画代码则为：

```
    this.setData({
      animationData:this.animation.export()
    })
```

这样动画就会按照我们指定的效果执行，需要注意的是，执行完 export()后，animation 对象指定的动画会被清除，如果需要再次使用，需要重新使用相应的动画方法来指定。

完整的 js 代码为：

```
Page({
  data:{
    animationData:{}
  },
  onShow:function(){
    // 页面显示
    this.animation = wx.createAnimation({
      duration: 500,
      timingFunction: 'ease-in-out',
    })
  },
  biggerFun:function(){
```

```
        this.animation.scale(2).step()
        this.setData({
          animationData:this.animation.export()
        })
    },

})
```

执行代码，并点击“放大”按钮，蓝色的 view 组件会通过一个渐变动画放大，最终如图 7.21 所示。

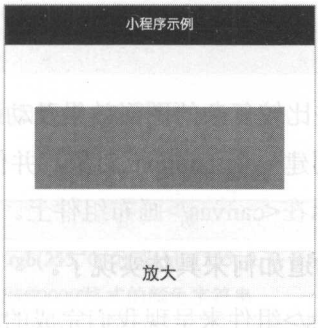


图 7.21

动画对象调用完一个指定的动画方法后会返回对象本身，因此可以使用链式调用来叠加其他动画，链式调用的每个动画都会以异步的方式执行，因此是同时执行的，直到用.step()方法来结束。在上述代码中，如果需要实现边放大边旋转，则可以把动画方法的语句改为：

```
this.animation.rotate(360).scale(2).step()
```

保存后，在点击“放大”按钮，会发现按钮会边旋转边放大，两个动画是同时进行的。

还可以设定动画队列，例如这样：

```
this.animation.rotate(360).scale(2).step()
this.animation.translate(100, 100).step()
```

先完成边放大边旋转的动画，然后再次点击按钮会进行之后的平移动画。

还可以在 setp()方法中传入与 wx.createAnimation()方法相同的参数，以实现改变当前动画一些参数，例如：

```
        this.animation.rotate(360).scale(2).step({
            duration: 3000,
            timingFunction: "ease-in-out",
        })
        this.animation.translate(100, 100).step()
```

wx.createAnimation 接口创建动画的方法就是这样，其他就是利用 this.animation 来根据需求使用不同的动画方法，读者可以自己动手多换几个动画参数和方法来实现不同动画效果，以加深对该接口用法的理解。

7.3.5 绘图

绘图通常用于实现一些比较复杂的图形效果及动画，小程序中，我们可以通过 wx.createContext()方法创建一个 context 对象，并使用 wx.drawCanvas 方法将 context 对象绘制的图形显示在<canvas/>画布组件上。

知道了大致流程，就知道如何来具体实现了。

首先，需要一个<canvas/>组件来呈现我们完成的动画，编写：

```
<canvas canvas-id="c1"/>
```

然后用 wx.createContext()接口来创建一个 context 对象，通常这类工作我们是放在 onReady 周期函数里，因为涉及到界面组件，放在 onReady 里是比较合适的，因此代码可以编写为：

```
onReady:function(){
    // 页面渲染完成
    var context = wx.createContext()
  },
```

之后就可以通过得到的 context 对象的方法来绘制需要的图形，比如想绘制一个矩形，context 的 rect 方法，该方法的参数如表 7.26 所示。

表 7.26　context 的 rect 方法参数列表

	参　数	类　型	范　围	说　明
1	*x*	Number		矩形路径左上角的 *x* 坐标
2	*y*	Number		矩形路径左上角的 *y* 坐标
3	width	Number		矩形路径的宽度
4	height	Number		矩形路径的高度

参数非常简单直观，前两个是坐标，后两个是宽高，无须过多解释，可以增加这样一条语句来画一个矩形路径：

```
context.rect(50, 60, 200, 100)
```

但这只是画完一个路径，如果需要显示出来，需要增加如下代码：

```
context.stroke()
```

这样就完成了一个矩形的绘制，但如果想更改矩形的边框颜色及填充色，就需要在描边之前设置边框颜色和填充色，需要用如下两条代码：

```
context.setStrokeStyle("#00ffff")
context.setFillStyle("#ff00ff")
```

这两个方法的参数说明如表 7.27 所示。

表 7.27　参数说明表

	参　数	类　型	范　围	说　明
1	color	String	rgb(255, 0, 0)'或'rgba(255, 0, 0, 0.6)'或'#ff0000'格式的颜色字符串	设置为填充样式的颜色

两个方法的参数用法和作用都一样。

其中 setFillStyle 用来设置填充色，setStrokeStyle 用来设置描边颜色，这两个方法都需要在进行描边和填充之前执行，然后再使用如下两条代码来进行描边和填充：

```
context.stroke()
context.fill()
```

这样就完成了矩形的绘制，但是，如果需要绘制在界面上，还需要通过 wx.drawCanvas 接口将这个 context 对象绑定到 canvas 组件上，以完成绘制图形的输出。语法为：

```
wx.drawCanvas(OBJECT)
```

其中 OBJECT 对象接收的参数如表 7.28 所示（官方文档提供）。

表 7.28　接口参数说明表

	参　数	类　型	必　填	说　明
1	canvasId	String	是	画布标识，传入<canvas/>的 cavas-id
2	actions	Array	是	绘图动作数组，由 wx.createContext 创建的 context，调用 getActions 方法导出绘图动作数组

续表

	参 数	类 型	必 填	说 明
3	reserve	Boolean	否	本次绘制是否接着上一次绘制，即 reserve 参数为 false，则在本次调用 drawCanvas 绘制之前 native 层应先清空画布再继续绘制；若 reserver 参数为 true，则保留当前画布上的内容，本次调用 drawCanvas 绘制的内容覆盖在上面，默认 false

这三个参数较为简单，这儿不再特别需要说明，因此直接使用这个接口来将绘制的矩形输出在组件上，代码如下：

```
    wx.drawCanvas({
      canvasId: 'cav1',
      actions: context.getActions()
})
```

可见，如果输出 context 绘制图形到 canvasId 参数指定的 canvas 组件上，需要在其 actions 属性中调用 context.getActions()方法来获得 context 对象上存储的绘图动作。

这样就实现了用 wx.createContext 创建一个绘图上下文 content 对象，并调用 content 绘图上下文对象的方法绘制矩形，再通过 wx.drawCanvas 接口将绘制的矩形输出在指定的 canvas 组件上，完整 js 文件的代码则为：

```
Page({
  onReady:function(){
    // 页面渲染完成
    var context = wx.createContext()
    // 先用 context 的方法绘制一个矩形
    context.rect(50, 60, 200, 100)
    //设置描边颜色
    context.setStrokeStyle("#00ffff")
    //设置填充颜色
    context.setFillStyle("#ff00ff")
    //对路径进行描边
    context.stroke()
    //对路径进行填充
    context.fill()

    wx.drawCanvas({
```

```
      canvasId: 'cav1',
      actions: context.getActions()
    })
  }
})
```

执行代码后，在页面上绘制的矩形效果如图 7.22 所示。

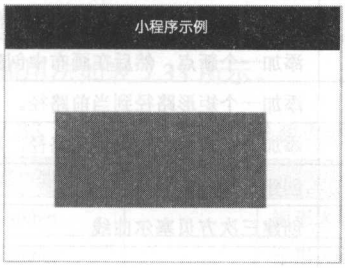


图 7.22

通过熟悉整个绘制流程，使用过 html5 规范里的 getContext()方法的同学对这个思路一定非常熟悉，其思路基本上是一样的，只是小程序重新做了一些封装，基本涵盖了 html5 规范里的大部分内容。

在官方文档中，提供了大量的 context 的方法，其用法基本一致。

context 对象的方法如表 7.29 所示。

表 7.29　context 对象的方法表

	方　法	参数	说　明
1	getActions	无	获取当前 context 上存储的绘图动作
2	clearActions	无	清空当前的存储绘图动作
3	变形		
4	scale		对横纵坐标进行缩放
5	rotate		对坐标轴进行顺时针旋转
6	translate		对坐标原点进行缩放
7	save	无	保存当前坐标轴的缩放、旋转、平移信息
8	restore	无	恢复之前保存过的坐标轴的缩放、旋转、平移信息
9	绘制		
10	clearRect		在给定的矩形区域内，清除画布上的像素
11	fillText		在画布上绘制被填充的文本
12	drawImage		在画布上绘制图像
13	fill	无	对当前路径进行填充

续表

	方　法	参数	说　明
14	stroke	无	对当前路径进行描边
15	路径		
16	beginPath	无	开始一个路径
17	closePath	无	关闭一个路径
18	moveTo		把路径移动到画布中的指定点，但不创建线条。
19	lineTo		添加一个新点，然后在画布中创建从该点到最后指定点的线条。
20	rect		添加一个矩形路径到当前路径。
21	arc		添加一个弧形路径到当前路径，顺时针绘制。
22	quadraticCurveTo		创建二次方贝塞尔曲线
23	bezierCurveTo		创建三次方贝塞尔曲线
24	样式		
25	setFillStyle		设置填充样式
26	setStrokeStyle		设置线条样式
27	setGlobalAlpha		设置全局画笔透明度
28	setShadow		设置阴影
29	setFontSize		设置字体大小
30	setLineCap		设置线条端点的样式
31	setLineJoin		设置两线相交处的样式
32	setLineWidth		设置线条宽度
33	setMiterLimit		设置最大倾斜

其中每个方法的参数和用法分别见表 7.30 所示。

表 7.30　scale 方法的参数表

	参　数	类　型	范　围	说　明
1	scaleWidth	Number	1 = 100%，0.5 = 50%，2 = 200%，依次类推	横坐标缩放的倍数
2	scaleHeight	Number	1 = 100%，0.5 = 50%，2 = 200%，依次类推	纵坐标轴缩放的倍数

调用该方法，其之前创建的路径其横纵坐标会被缩放，若不使用 clearActions 方法清除动作，则多次调用会被累加缩放，其倍数相乘。

rotate 方法的参数和用法如表 7.31 所示。

表 7.31　rotate 方法的参数说明表

	参　数	类　型	范　围	说　明
1	rotate	Number	degrees * Math.PI/180；degrees 范围为 0~360	旋转角度，以弧度计

其多次被调用的情况与上述的 scale 方法一致。

translate 方法的参数和用法如表 7.32 所示。

表 7.32　translate 方法的参数说明表

	参　数	类　型	范　围	说　明
1	*x*	Number		水平坐标平移量
2	*y*	Number		竖直坐标平移量

clearRect 方法的参数和用法如表 7.33 所示。

表 7.33　clearRect 方法的参数说明表

	参　数	类　型	范　围	说　明
1	*x*	Number		矩形区域左上角的 *x* 坐标
2	*y*	Number		矩形区域左上角的 *y* 坐标
3	width	Number		矩形区域的宽度
4	height	Number		矩形区域的高度

drawImage 方法的参数和用法如表 7.34 所示。

表 7.34　drawImage 方法的参数说明表

	参　数	类　型	范　围	说　明
1	imageResource	String	通过 chooseImage 得到一个文件路径或者一个项目目录内的图片	所要绘制的图片资源
2	*x*	Number		图像左上角的 *x* 坐标
3	*y*	Number		图像左上角的 *y* 坐标
4	width	Number		图像宽度
5	height	Number		图像高度

fillText 方法的参数和用法如表 7.35 所示。

表 7.35　fillText 方法的参数说明表

	参　数	类　型	说　明
1	text	String	在画布上输出的文本
2	*x*	Number	绘制文本的左上角 *x* 坐标位置
3	*y*	Number	绘制文本的左上角 *y* 坐标位置

beginPath 方法用于开始创建一个路径，需要调用 fill 或者 stroke 才会使用路径进行填充或描边。其 setFillStyle、setStrokeStyle 设置后者会覆盖前者。

closePage 方法用于关闭一个路径。

moveTo 方法用于把路径移动到画布中指定的点，其参数和用法如表 7.36 所示。

表 7.36 moveTo 方法参数说明表

	参 数	类 型	说 明
1	*x*	Number	目标位置的 *x* 坐标
2	*y*	Number	目标位置的 *y* 坐标

lineTo 方法用于在当前位置添加一个新点，然后在画布中创建从该点到最后指定点的路径。其参数就是 *x* 和 *y* 坐标，指目标位置的 *x*，*y* 坐标。

rect 方法用于添加一个矩形路径到当前路径，其用法前面已经做过介绍。

arc 方法用于添加一个弧形路径到当前路径，顺时针绘制。其参数和用法如表 7.37 所示。

表 7.37 arc 方法参数说明表

	参 数	类 型	范 围	说 明
1	*x*	Number		矩形路径左上角的 *x* 坐标
2	*y*	Number		矩形路径左上角的 *y* 坐标
3	radius	Number		矩形路径的宽度
4	startAngle	Number	弧度, 0 到 2pi	起始弧度
5	sweepAngle	Number	弧度, 0 到 2pi	从起始弧度开始，扫过的弧度

quadraticCurveTo 用于创建二次贝塞尔曲线路径，其参数及用法如表 7.38 所示。

表 7.38 quadraticCurveTo 方法参数说明表

	参 数	类 型	说 明
1	cpx	Number	贝塞尔控制点的 *x* 坐标
2	cpy	Number	贝塞尔控制点的 *y* 坐标
3	*x*	Number	结束点的 *x* 坐标
4	*y*	Number	结束点的 *y* 坐标

bezierCurveTo 方法用于创建三次方贝塞尔曲线路径，其参数及用法如表 7.39 所示。

表 7.39　bezierCurveTo 方法参数说明表

	参　数	类　型	说　明
1	cp1x	Number	第一个贝塞尔控制点的 x 坐标
2	cp1y	Number	第一个贝塞尔控制点的 y 坐标
3	cp2x	Number	第二个贝塞尔控制点的 x 坐标
4	cp2y	Number	第二个贝塞尔控制点的 y 坐标
5	x	Number	结束点的 x 坐标
6	y	Number	结束点的 y 坐标

setFillStyle 方法用来设置纯色填充，其用法前面介绍过了。

setStrokeStyle 方法用于设置纯色描边，之前的例子也已经用到过。

setGlobalAlpha 方法用来设置全局画笔透明度，其参数和用法如表 7.40 所示。

表 7.40　setGlobalAlpha 参数说明表

	参　数	类　型	范　围	说　明
1	alpha	Number	0~1	透明度，0 表示完全透明，1 表示完全不透明

setShadow 方法用来设置阴影，其参数与用法如表 7.41 所示。

表 7.41　setShadow 方法参数说明表

	参数	类型	范　围	说　明
1	offsetX	Number		阴影相对于形状在水平方向的偏移
2	offsetY	Number		阴影相对于形状在竖直方向的偏移
3	blur	Number	0~100	阴影的模糊级别，数值越大越模糊
4	color	Color	rgb(255, 0, 0)'或'rgba(255, 0, 0, 0.6)'或'#ff0000'格式的颜色字符串	阴影的颜色

setFontSize 方法用来设置字体的字号，其参数与用法如表 7.42 所示。

表 7.42　setFontSize 参数说明表

	参　数	类　型	说　明
1	fontSize	Number	字体的字号

setLineWidth 方法用来设置线条的宽度，其参数与用法如表 7.43 所示。

表 7.43 setLineWidth 参数说明表

	参 数	类 型	说 明
1	lineWidth	Number	线条的宽度

setLineCap 方法用来设置线条的结束端点样式，其参数与方法如表 7.44 所示。

表 7.44 setLineCap 方法参数说明表

	参 数	类 型	范 围	说 明
1	lineCap	String	butt'、'round'、'square'	线条的结束端点样式

setLineJoin 方法设置两条线相交时，所创建的拐角类型，如表 7.45 所示。

表 7.45 setLineJoin 方法参数说明表

	参 数	类 型	范 围	说 明
1	lineJoin	String	bevel'、'round'、'miter'	两条线相交时，所创建的拐角类型

setMiterLimit 方法设置最大斜接长度，斜接长度指的是在两条线交汇处内角和外角之间的距离。当 setLineJoin 为 miter 时才有效。超过最大倾斜长度的，连接处将以 lineJoin 为 bevel 来显示，其参数和用法如表 7.46 所示。

表 7.46 setMiterLimit 方法参数说明表

	参 数	类 型	说 明
1	miterLimit	Number	最大斜接长度

这些方法的用法总体上来说和之前介绍的思路一样，只是每个方法可能会有自己的用法特点，相信这一点对于有图形编程基础的同学来说会非常的简单。对于运用 canvas 来绘制图形来说，图形编程基础也是需要的，因此，这里就不做更多的详细解释了，只要对创建 canvas 动画的总体思路有了解即可，对此感兴趣的读者也可以自己通过实例多尝试去学习，这样掌握的也更牢固一些。

除了在 canvas 上显示绘图的内容，也可以将 canvas 上输出的内容生成图像文件，可以使用如下接口：

```
wx.canvasToTempFilePath(OBJECT)
```

其 OBJECT 对象接受如表 7.47 所示参数。

表 7.47　接口参数列表

	参　数	类　型	必　填	说　明
1	canvasId	String	是	画布标识，传入<canvas/> 的 cavas-id

只有一个 canvasId 参数，指定需要生成图像文件的 canvas 组件的 cavas-id 即可。

7.3.6　其他

有时候需要在特殊的情况下主动收起用户的输入法键盘，此时可以使用接口：

```
wx.hideKeyboard()
```

在前面章节介绍页面事件时，提到过 onPullDownRefresh 方法用来实现页面的下拉刷新，如果下拉数据内容已经加载完了，就需要停止下拉刷新，此时就可以使用以下接口实现：

```
wx.stopPullDownRefresh()
```

7.4　手把手教你做 Demo——小地图

来运用本章学习的制作一个简单的地图程序，熟悉一下本章学习的知识。

（1）首先在 wxml 文件中编写如下代码：

```
<view >
    <map id="theMap" class="theMap"
    show-location />
    <button        class="vi"        bindtap="getCenterLocationFun"
animation="{{animationData}}">
        获取坐标
    </button>
    <button  class="vi"  bindtap="moveToLocationFun"  animation="
{{animationData}}">
        定位到当前位置
    </button>
</view>
```

这里注意，我们在每一个按钮上都增加了 animation 属性，目的是为按钮增加一个从大变小的动画效果。

可能有读者要问，为什么不给 map 组件增加 animation 属性呢？

笔者也考虑过为 map 组件外的 view 组件增加 animation 属性，并实现相同的效果，但是，在开发者工具中可以看到效果，却无法在手机中看到效果，这一块可能会和 map 组件有一些冲突，无法实现。

因此先在 button 按钮上实现这个效果。

（2）紧接着在 wxss 文件中编写如下样式代码：

```
.theMap{
    width:100%;
    height:800rpx;
}

.vi{
    transform: scale(4);
}
```

很简单，这里就不解释了。

（3）然后在 js 文件中编写如下代码：

```
Page({
data:{
    animationData:{}
  },
  onReady:function(){
    // 生命周期函数--监听页面初次渲染完成

    this.mapContext = wx.createMapContext('theMap')
  },
  onShow:function(){
    // 生命周期函数--监听页面显示
    var animation = wx.createAnimation({
      duration: 700,
        timingFunction: 'ease',
    })
```

```
    this.animation = animation

    animation.scale(1).step()

    this.setData({
      animationData:animation.export()
    })
  },

  /**
   * 获取当前地图中心的经纬度
   */
  getCenterLocationFun: function () {
    var _self=this;
    //获得网络状态，来判断是否联网
    wx.getNetworkType({
      success: function(res) {
        //如果 networkTyp 值为 none，则说明手机处于断网状态
        if(res.networkType=='none'){
          wx.showToast({
            title: '没有网络连接，sorry',
            icon: 'success',
            duration: 2000
          })
        }else{
          //否则处于联网状态
          _self.mapContext.getCenterLocation({
            success: function(res){
              wx.showToast({
                title: '获取到经纬度:'+res.longitude+','+res.
latitude,
                icon: 'success',
                duration: 2000
              })
            }
          })
        }
      }
```

```
    })

  },

  /**
   * 将当前地图移动到定位位置
   */
  moveToLocationFun: function () {
    var _self=this;
    wx.getNetworkType({
        success: function(res) {
          //与上述同理
          if(res.networkType=='none'){
              wx.showToast({
                title: '没有网络连接，sorry',
                icon: 'success',
                duration: 2000
              })
          }else{
              _self.mapContext.moveToLocation()
          }
        }
    })

  }
})
```

代码主要实现了三个功能。

（1）在 onShow 周期函数方法中，实现了按钮动效，主要代码为：

```
var animation = wx.createAnimation({
     duration: 700,
       timingFunction: 'ease',
   })

   this.animation = animation

   animation.scale(1).step()

   this.setData({
```

```
        animationData:animation.export()
    })
```

（2）在 getCenterLocationFun 方法中，通过 wx.getNetworkType 接口获得了网络状态，通过判断网络状态 res.networkType 是否为 none 来判断是否有网络连接，在有网络状态下通过接口 mapContext.getCenterLocation 获得当前地图中心的经纬度，并调用接口 wx.showToast 来显示获得的经纬度。

（3）使用同样的方法来判断是否联网，然后再通过接口 mapContext.moveToLocation 来移动到定位位置。

可见通过这个简单的实例，将几个比较重要的接口练习了一下，运行后效果如图 7.23 所示。

如果读者将手机网络切换为飞行模式，再点击任意按钮，则效果如图 7.24 所示。

图 7.23

图 7.24

7.5　本章要点总结

这一章了解如何运用有关地图的 API，学习了如何实现一些与地理位置有关的功能。在开发与地理位置有关的功能时，使用 Map 组件和相关 API 进行关联开发的思路和方法需要都掌握，相信读者在开发这类应用时都能得心应手。

通过设备有关 API，我们可以很方便地利用手机硬件实现一些有趣的应用，本章也通过手把手环节，通过获得网络状态的 API 实现了判断用户设备是否联网。

一些关于用户界面的 API 可以使得我们的应用体验更加友好。

后面又学习了小程序提供的用来实现动画的 API，以及与组件绑定的方法，小程序用这些方法也在一定程度上降低了动画开发的难度。

绘图方面的 API 由于目前小程序不允许用来开发游戏，因此更多的使用场景应该是用来绘制一些饼状图之类的图形，以及一些特殊场景的不规则图形。使用的基本方法也在本章做了一些介绍。

对于本章的内容，相信读者已经基本掌握，也希望读者能够通过多练习本章实例，来加深对相关内容的了解。

第 8 章

小程序 API（3）：开放接口

本章我们了解下微信小程序的开放接口以及它的使用方法。

8.1 登录

登录是我们开发小程序必备的一个功能，通过登录我们可以标识用户，为用户提供持续性的服务，这一点的重要性无须多说，与在公众号里实现登录类似，在小程序中实现登录，需要调用 wx.login 接口来实现，并需要在自己的服务器上维护用户的登录状态信息。

8.1.1 wx.login(OBJECT)

该接口用来实现用户登录，并获得登录凭证 code，使用该 code 通过第三方服务器获取用户唯一标识，即 openid，以及会话密匙，即 session_key。

其接受的参数如表 8.1 所示。

表 8.1 接口参数说明列表

参数名	类 型	必 填	说 明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

从表 8.1 可见，其只接受普通的 success，fail 和 complete 参数，也就是说无须指定特殊的参数，因此使用如下代码即可实现登录：

```
wx.login({
  success: function(res) {
    console.log('登录凭证',res.code)
  }
});
```

用该接口实现登录仅仅需要这行代码。

从中可以看到通过 res 参数对象就可以取得登录凭证 code，其 res 对象的参数内容如表 8.2 所示。

表 8.2　接口成功返回参数说明表

参数名	类　型	说　明
errMsg	String	调用结果
code	String	用户允许登录后，回调内容会带上 code（有效期五分钟），开发者需要将 code 发送到开发者服务器后台，使用 code 换取 session_key api，将 code 换成 openid 和 session_key

登录功能非常简单，但如果我们需要通过获得的登录凭证 code，来获得 session_key，则还需要通过接口：

https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code

其接口参数如表 8.3 所示。

表 8.3　接口参数说明列表

参　数	必　填	说　明
appid	是	小程序唯一标识
secret	是	小程序的 app secret
js_code	是	登录时获取的 code
grant_type	是	填写为 authorization_code

按照官方文档的说明，该接口返回情况如下：

```
//正常返回的 JSON 数据包
{
    "openid": "OPENID",
    "session_key": "SESSIONKEY"
```

```
}
//错误时返回 JSON 数据包(示例为 Code 无效)
{
    "errcode": 40029,
    "errmsg": "invalid code"
}
```

这里需要注意的是，我们并不能在客户端通过这个接口来获得 session_key，这一点从技术上来说也比较难实现，因为在小程序中，非“request 合法域名”中填写的域名，我们在小程序里是无法用 request 接口来访问的，因此无法通过客户端来获得 session_key，这也不符合微信官方文档中所述的规范，而正确的做法应该是我们通过 request 接口将登录凭证 code 发送到我们自己的服务器上，然后在服务器上再通过上述 https 接口来进一步获得 session_key，而按照官方文档的描述，我们也不应该直接将 session_key 回传客户端使用，而是自行根据 session_key 和 openid 再生成一个 3rd_session。

对于 3rd_session 的生成规则，官方也做了一些建议：

（1）长度达到 16byte；

（2）避免当前使用然后取随机数的方法，而是采用操作系统提供的真正的随机数，例如 Linux 下读取/dev/urandom 设备；

（3）设置一定的有效时间，过期后失效。

再将这个 3rd_session 传回客户端，客户端将该 3rd_session 存储在 storage 中，登录失效时再将该 3rd_session 通过 wx.request()接口向自己的服务器请求，服务器再从 session 存储中查找合法的 session_key 和 openid。

该逻辑其实简单来说，最合理的方案为，用户登录后向服务器发送 code，并在服务器上通过该 code 和其他一些参数请求接口：

https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code

获得 openid 和 session_key，再通过这两个值生成一个 3rd_session，存储在服务端 session，并返回客户端，客户端在之后需要获得 openid 和 session_key 时，都通过 wx.request()接口来访问自己的服务器，通过传入 3rd_session，通过服务器来判断用户是否还是登陆状态。

按照这个思路再看官方提供的流程图，就会非常清晰明了，如图 8.1 所示。

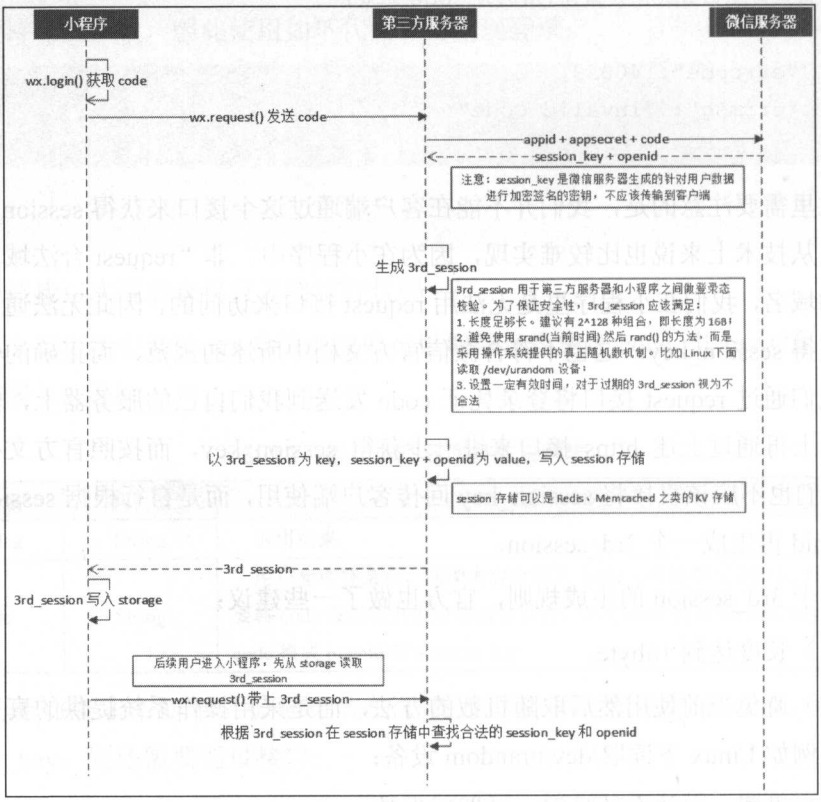


图 8.1

图 8.1 需要从左上角看到右上角，再逐渐往下走到左下角，结合我们的一些分析，就能比较清晰的了解了。

由于涉及到服务器端的实现，不在本书讨论的范围内，对此就不再展开阐述。明白了思路，读者可以自行编写相应逻辑，其实现其实主要还是根据自己的需求来定。

8.1.2 wx.checkSession(OBJECT)

此外，对于登录，微信小程序还提供了接口：

```
wx.checkSession(OBJECT)
```

用来检查登录状态是否过期，其 OBJECT 对象接受如表 8.4 所示的参数。

表 8.4　接口参数说明列表

参数名	类　型	必　填	说　明
success	Function	否	接口调用成功的回调函数，登陆态未过期
fail	Function	否	接口调用失败的回调函数，登陆态已过期
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

可见该参数除了常规的 success、fail 和 complete 方法参数，没有其他特别的参数。

其用法其实也比较特殊，如果登录状态未过期，会调用 success 方法，如果过期了，则会调用 fail 方法。

实现代码如下：

```
wx.checkSession({
  success: function(){
//登录态未过期
console.log('登录状态未过期 ')
  },
  fail: function(){
//登录态过期
console.log('登录状态已过期，应该进行登录')
  }
})
```

8.1.3　用户数据的签名验证和加解密

微信对于一部分比较敏感的用户数据是进行加密传输的，例如包含 openid 的用户数据，这个情况再我们使用 wx.getUserInfo 接口时就会碰到。因此，如果我们需要使用这些数据进数据校验，则需要 session_key、rawData、signature 这些信息，其中：

（1）session_key 是在调用 wx.login 实现登录时，通过一个 https 请求地址获得，即：

https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code

该接口正常的返回 Json 为：

```
{
    "openid": "OPENID",
    "session_key": "SESSIONKEY"
}
```

其中的 session_key 即我们解密需要用到的数据。

前面章节讲到过，这一步请求是需要在我们自己的服务器上完成的。

（2）rawData、signature 数据可以通过 wx.getUserInfo 接口获得，在后续介绍 wx.getUserInfo 接口时便可以了解到，通过该接口，可以获得如图 8.2 所示的信息。

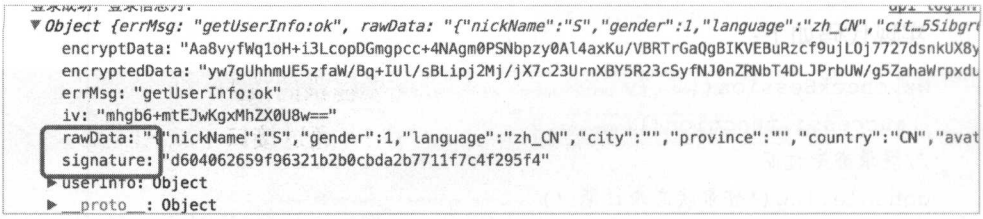


图 8.2

从图中可见，已获得需要的数据。其中，signature = sha1(rawData + session_key)。

（3）有了 signature、rawData 和 session_key，就可以进行数据完整性的校验了，方法就是将 signature、rawData 发送到自己的服务器中，假设在服务器中也得到了 session_key，就可以通过 sha1(rawData + session_key)的方法计算出 signature，与通过 wx.getUserInfo 获得的 signature 做比对，如果相等，则数据完整正确，否则就是不正确的。

如果需要对这部分数据进行解密，官方文档提供了比较详细的解密算法，但是在使用的过程中，一般不需要关心这个比较复杂的解密算法，官方已经提供了多数服务端语言的解密示例，直接使用即可。下面以官方文档提供的 node.js 语言版本来讲解一下使用方法。

从官方文档的 API“开放接口-登录-签名加密-加密数据解密算法”部分，可以找到“微信官方提供了多种编程语言的示例代码（点击下载）”的文字部分，点击下载链接，即可下载一个实例压缩包，其中有 c++、Node、PHP、Python 四种语言的实例，这里选择 Node，打开 demo.js 文件，为了便于理解，去掉干扰内容，将代码简化为：

```
var WXBizDataCrypt = require('./WXBizDataCrypt')

var appId = '{appId}'
var sessionKey = '{sessionKey}'
var encryptedData = '{encryptedData}'
var iv = '{iv}'

var pc = new WXBizDataCrypt(appId, sessionKey)

var data = pc.decryptData(encryptedData , iv)
```

其中 var WXBizDataCrypt = require('./WXBizDataCrypt')是引入了实现解密的具体方法，其代码实现并不复杂，有兴趣的读者可以自行阅读。

剩下的 sessionKey、encryptedData、iv，都是需要提供的数据，得到 sessionKey 的方法通过之前的讲解已经清楚，encryptedData 和 iv 通过接口 wx.getUserInfo 即可获取，如图 8.3 所示画方框处。

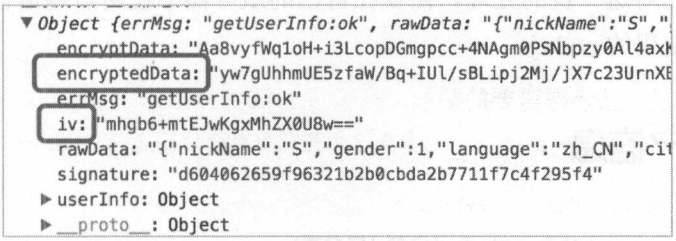


图 8.3

有了这些数据，通过以下代码：

```
var pc = new WXBizDataCrypt(appId, sessionKey)

var data = pc.decryptData(encryptedData , iv)
```

可以得到解密后的数据，即 data 里的数据，示例代码也给出了打印出 data 的注释内容：

```
data = {
  "openId": "OPENID",
  "nickName": "Band",
  "gender": 1,
  "language": "zh_CN",
  "city": "Guangzhou",
```

```
    "province": "Guangdong",
    "country": "CN",
    "avatarUrl":
"http://wx.qlogo.cn/mmopen/vi_32/aSKcBBPpibyKNicHNTMM0qJVh8Kjgiak2AH
Wr8MHM4WgMEm7GFhsf8OYrySdbvAMvTsw3mo8ibKicsnfN5pRjl1p8HQ/0",
    "unionId": "ocMvos6NjeKLIBqg5Mr9QjxrP1FA",
    "watermark": {
      "timestamp": 1477314187,
      "appid": "wx4f4bc4dec97d474b"
    }
  }
```

可以看到，获得了比较敏感的 openid，以及 watermark 参数，watermark 参数官方称之为数据水印，其中包含了 appid 和 timestamp，也可以校验 appid 是不是自身 appid，并且根据 timestamp 校验数据时效性。

从这一部分内容可以得知，对加密数据的校验和解密，都是需要在服务器中进行的，因此对这一块的处理，需要在自己的服务器中使用后端语言实现，例如本例讲解中使用的 node.js。

8.2 用户信息

8.2.1 wx.getUserInfo(OBJECT)

通过 wx.login 接口，可以实现用户的登录，并在自己的服务器上管理登录状态，实现了登录以后，就可以使用 wx.getUserInfo 接口来获得用户的信息了。

该接口的 OBJECT 对象接收的参数如表 8.5 所示。

表 8.5 接口参数说明列表

参数名	类 型	必 填	说 明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

可见其参数也只接受默认的 success、fail 和 complete 方法，因此，在使用前使用 wx.login 登录过后，就可以使用本接口获得用户的数据了，用户数据会通过 success 方法的参数对象传入，返回参数如表 8.6 所示。

表 8.6　接口成功返回参数说明表

参　数	类　型	说　明
userInfo	OBJECT	用户信息对象，不包含 openid 等敏感信息
rawData	String	不包括敏感信息的原始数据字符串，用于计算签名。
signature	String	使用 sha1(rawData + sessionkey) 得到字符串，用于校验用户信息。
encryptedData	String	包括敏感数据在内的完整用户信息的加密数据，详细见加密数据解密算法
iv	String	加密算法的初始向量，详细见加密数据解密算法

通过一段简单的代码，即可实现一个简单的登录功能，并获得用户的数据：

```
wx.login({
  success: function(res) {
    if (res.code) {
      console.log('登录成功: ',res)
      //发起网络请求
      wx.getUserInfo({
      success: function(r) {
        console.log('登录成功，登录信息为:',r);
      }
    })
    } else {
      console.log('获取用户登录态失败! ' + res.errMsg)
    }
  }
});
```

其中 wx.login 是登录，在其 seccess 方法中，继续调用 wx.getUserInfo 就可以获得用户的登录数据，运行这段代码后，会在控制台打印出如图 8.4 所示的用户数据。

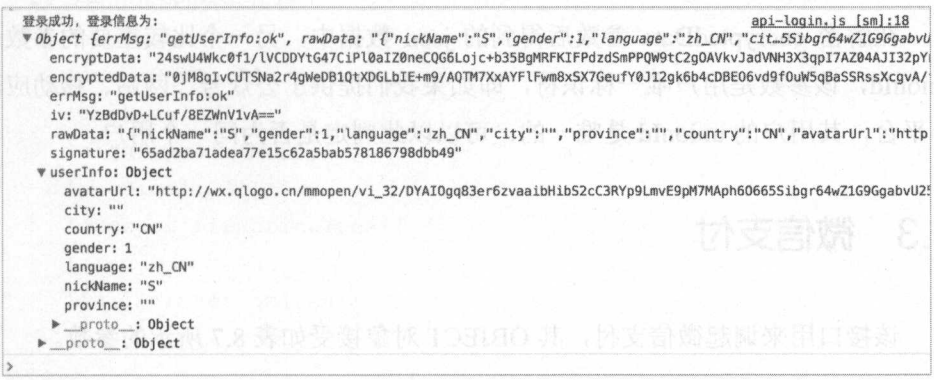


图 8.4

对比表 8.6，可以更直观地理解得到的个人数据信息，其中 userInfo 只包含一些基本的用户信息，涉及到 openid 等敏感的信息，则不在 userInfo 字段中，而这些敏感数据是存放在 encryptedData 字段中的，并且这部分数据是经过加密的，如果需要这些数据，则需要对该字段进行解密，解密的方法在前面章节中做过介绍，从中可知，如果需要解密，标准的做法是需要将 encryptedData 数据传到自己的服务端，再通过相应代码来解密的，解密后，可以得到如下的 json 数据：

```
{
    "openId": "OPENID",
    "nickName": "NICKNAME",
    "gender": GENDER,
    "city": "CITY",
    "province": "PROVINCE",
    "country": "COUNTRY",
    "avatarUrl": "AVATARURL",
    "unionId": "UNIONID",
    "watermark":
    {
        "appid":"APPID",
    "timestamp":TIMESTAMP
    }
}
```

可见，这个数据中包含了 openId 等敏感的数据。

8.2.2 UnionID 机制

在解密 encryptedData 参数后得到的 json 数据中，另一个比较重要的参数是 unionId，该参数是用户唯一标识符，即如果我们提供了公众号、网站、移动应用等平台，其用户的 unionId 是唯一的，可以以此判定是否为同一个用户。

8.3 微信支付

该接口用来调起微信支付，其 OBJECT 对象接受如表 8.7 所示的参数。

表 8.7　接口参数说明列表

参　数	类　型	必　填	说　　明
timeStamp	String	是	时间戳从 1970 年 1 月 1 日 00:00:00 至今的秒数,即当前的时间
nonceStr	String	是	随机字符串，长度为 32 个字符以下。
package	String	是	统一下单接口返回的 prepay_id 参数值，提交格式如：prepay_id=*
signType	String	是	签名算法，暂支持 MD5
paySign	String	是	签名,具体签名方案参见微信公众号支付帮助文档;
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

这些参数的用法涉及微信支付流程，其方法与微信公众号的支付逻辑是一致的，这一部分内容超出本书内容，暂不详解，感兴趣的用户需要自行阅读官方文档与微信支付相关内容，并需要实现相应的服务器端逻辑。

该接口调用后返回如表 8.8 所示结果。

表 8.8　接口成功返回参数说明表

回调类型	errMsg	说　　明
success	requestPayment:ok	调用支付成功
fail	requestPayment:fail cancel	用户取消支付
fail	requestPayment:fail (detail message)	调用支付失败，其中 detail message 为后台返回的详细失败原因

接口的调用方法如下：

```
wx.requestPayment({
   'timeStamp': '',
   'nonceStr': '',
   'package': '',
   'signType': 'MD5',
   'paySign': '',
   'success':function(res){
   },
   'fail':function(res){
   }
})
```

8.4 客服消息

8.4.1 接收消息和事件

在 wxml 页面中使用<contact-button/>标签后，会在小程序中出现一个客服消息图标。

例如如果在 wxml 文件中用如下代码：

```
<contact-button
        type="default-dark"
        size="40"
        session-from="weapp"
        >
    </contact-button>
```

运行后则会在页面中看到如图 8.5 所示的图标。

图 8.5

请注意，这里的 type 属性如果指定为 default-light，在白色背景页面上会因为图标的颜色也为白色，而看不到效果。

在手机中，如果用户点击该图标，则会进入一个客服对话页面，如图 8.6 所示。

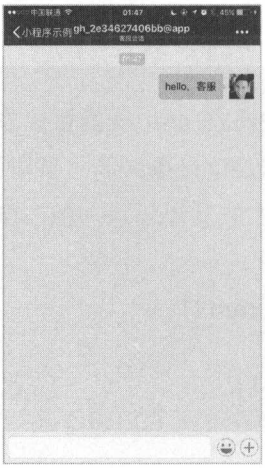


图 8.6

微信服务器在接收用户发送的消息后，会向开发者填写的 URL 发送 POST 请求，开发者在服务端收到消息后，可以使用发送客服消息接口进行回复。

需要注意的是，微信服务器向开发者服务器发送消息后，如果 5 秒内收不到响应会断掉连接，并重新发起请求，一共会重试三次。在调试中，如果用户无法收到消息，可以检查消息处理是否超时。

在收到微信服务器收到的客户消息后，可以做如下两种处理：

（1）直接回复 success（推荐方式）。

（2）直接回复空串（指字节长度为 0 的空字符串，而不是结构体中 content 字段的内容为空）。

以上两种方式都可以避免微信服务器发起重试，如果 5 秒内没有得到服务器的任何回复，或者我们的服务器回复了异常的数据，微信都会在客服对话框中提示用户：“该小程序客服暂时无法提供服务，请稍后再试”。

如果读者需要增强传输消息的安全性，则可以参照前面章节的加密解密方法。

按照官方文档提供的内容，用户在客服会话中发送消息，我们的服务器会收到如下的数据内容：

XML 格式：

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>1482048670</CreateTime>
  <MsgType><![CDATA[text]]></MsgType>
  <Content><![CDATA[this is a test]]></Content>
  <MsgId>1234567890123456</MsgId>
</xml>
```

JSON 格式：

```
{
    "ToUserName": "toUser",
    "FromUserName": "fromUser",
    "CreateTime": 1482048670,
    "MsgType": "text",
    "Content": "this is a test",
    "MsgId": 1234567890123456
```

```
}
```

做过公众号开发的读者对这个消息格式一定也不会陌生。

其参数说明如表 8.9 所示。

表 8.9　消息格式参数说明表

参　数	说　明
ToUserName	小程序的原始 ID
FromUserName	发送者的 openid
CreateTime	消息创建时间(整型)
MsgType	text
Content	文本消息内容
MsgId	消息 id，64 位整型

如果用户在客服对话里发送图片消息，则我们会收到如下的数据，XML 格式：

```
<xml>
    <ToUserName><![CDATA[toUser]]></ToUserName>
    <FromUserName><![CDATA[fromUser]]></FromUserName>
    <CreateTime>1482048670</CreateTime>
    <MsgType><![CDATA[image]]></MsgType>
    <PicUrl><![CDATA[this is a url]]></PicUrl>
    <MediaId><![CDATA[media_id]]></MediaId>
    <MsgId>1234567890123456</MsgId>
</xml>
```

JSON 格式：

```
{
    "ToUserName": "toUser",
    "FromUserName": "fromUser",
    "CreateTime": 1482048670,
    "MsgType": "image",
    "PicUrl": "this is a url",
    "MediaId": "media_id",
    "MsgId": 1234567890123456
}
```

其参数说明如表 8.10 所示。

表 8.10　消息格式参数说明表

参　数	说　明
ToUserName	小程序的原始 ID
FromUserName	发送者的 openid
CreateTime	消息创建时间(整型)
MsgType	image
PicUrl	图片链接（由系统生成）
MediaId	图片消息媒体 id，可以调用获取临时素材接口拉取数据
MsgId	消息 id，64 位整型

在用户进入客服对话时，微信会向我们的服务器发送如下的数据。

XML 格式：

```
<xml>
    <ToUserName><![CDATA[toUser]]></ToUserName>
    <FromUserName><![CDATA[fromUser]]></FromUserName>
    <CreateTime>1482048670</CreateTime>
    <MsgType><![CDATA[event]]></MsgType>
    <Event><![CDATA[user_enter_tempsession]]></Event>
    <SessionFrom><![CDATA[sessionFrom]]></SessionFrom>
</xml>
```

JSON 格式：

```
{
    "ToUserName": "toUser",
    "FromUserName": "fromUser",
    "CreateTime": 1482048670,
    "MsgType": "event",
    "Event": "user_enter_tempsession",
    "SessionFrom": "sessionFrom"
}
```

其参数说明如表 8.11 所示。

表 8.11　消息格式参数说明表

参　数	说　明
ToUserName	小程序的原始 ID
FromUserName	发送者的 openid
CreateTime	事件创建时间(整型)

续表

参　数	说　明
MsgType	Event
Event	事件类型，user_enter_tempsession
SessionFrom	开发者在客服会话按钮设置的 sessionFrom 参数

8.4.2　发送客服消息

从用户进入客服界面开始，就可以向用户推送客服消息了。

如果用户只是进入了客服界面，只能在进入后的一分钟内向用户推送一条消息。如果用户通过客服向我们发送了消息，则在接下来的 48 小时内，可以向用户推送三条客服消息，官方文档提供的内容表 8.12 所示。

表 8.12　客服消息限制表

用户动作	允许下发条数限制	下发时限
用户通过客服消息按钮进入会话	1 条	1 分钟
用户发送信息	3 条	48 小时

如果回复的消息条数达到上限，则会收到错误返回码，如表 8.13 所示。

表 8.13　错误码说明表

参　数	说　明
-1	系统繁忙，此时请开发者稍候再试
0	请求成功
40001	获取 access_token 时 AppSecret 错误，或者 access_token 无效。请开发者认真比对 AppSecret 的正确性，或查看是否正在为恰当的小程序调用接口
40002	不合法的凭证类型
40003	不合法的 OpenID，请开发者确认 OpenID 否是其他小程序的 OpenID
45015	回复时间超过限制
45047	客服接口下行条数超过上限
48001	api 功能未授权，请确认小程序已获得该接口

向用户发送客服消息的接口为：

https://api.weixin.qq.com/cgi-bin/message/custom/send?access_token=ACCESS_TOKEN

接口调用方式为 POST，其发送文本消息的 json 数据包内容格式为：

```
{
    "touser":"OPENID",
    "msgtype":"text",
    "text":
    {
        "content":"Hello World"
    }
}
```

发送图片消息的 json 数据包内容格式为：

```
{
    "touser":"OPENID",
    "msgtype":"image",
    "image":
    {
      "media_id":"MEDIA_ID"
    }
}
```

其中参数的含义如表 8.14 所示。

表 8.14　接口参数说明表

参　数	是否必须	说　明
access_token	是	调用接口凭证
touser	是	普通用户，openid
msgtype	是	消息类型，文本为 text，图片为 image
content	是	文本消息内容
media_id	是	发送的图片的媒体 ID，通过新增素材接口上传图片文件获得

调用接口后错误返回码则可以参照表 8.13 所示。

8.4.3　临时素材接口

我们还可以在小程序中获取客服消息内收到的临时素材，写本书时，暂时还只支持图片，请求的接口为：

https://api.weixin.qq.com/cgi-bin/media/get?access_token=ACCESS_TOKEN&media_id=MEDIA_ID

接口的参数说明如表 8.15 所示。

表 8.15　接口参数说明表

参　数	是否必须	说　明
access_token	是	调用接口凭证
media_id	是	媒体文件 ID

其中的 media_id 即是微信服务器向我们发送的用户客服消息 JSON 或者 XML 数据包里的参数。

错误情况下的返回 JSON 数据包示例如下（示例为无效媒体 ID 错误）：

```
{
  "errcode":40007,
  "errmsg":"invalid media_id"
}
```

其错误返回码可以参照 8.4.2 章节的表 8.13 所示。

如果需要将媒体文件临时上传到微信服务器上，则可以通过 POST/FORM 方式请求该接口，写本书时，只支持图片，该接口为：

https://api.weixin.qq.com/cgi-bin/media/upload?access_token=ACCESS_TOKEN&type=TYPE

接口参数含义如表 8.16 所示。

表 8.16　接口参数说明表

参　数	是否必须	说　明
access_token	是	调用接口凭证
type	是	image
media	是	form-data 中媒体文件标识，有 filename、filelength、content-type 等信息

如果上传成功，则接口返回的 json 内容为：

```
{
  "type":"TYPE",
  "media_id":"MEDIA_ID",
  "created_at":123456789
}
```

其内容含义如表 8.17 所示。

表 8.17　接口返回参数说明表

参　数	描　述
type	image
media_id	媒体文件上传后，获取标识
created_at	媒体文件上传时间戳

如果接口调用出错，则依然返回如下 json 内容：

```
{
  "errcode":40004,
  "errmsg":"invalid media type"
}
```

8.5　分享

小程序提供了分享功能，但分享目前只能是分享到群里，或者分享给朋友，不能分享到朋友圈。

在小程序中实现分享非常简单，只需在 Page 函数中定义 onShareApp Message 函数，实现方法类似于小程序的生命周期函数，但是也有区别，不同之处是 onShareAppMessage 函数需要 return 一个对象，以自定义分享内容。自定义字段内容如表 8.18 所示。

表 8.18　自定义分享字段表

	字　段	说　明	默 认 值
1	title	分享标题	当前小程序名称
2	desc	分享描述	当前小程序名称
3	path	分享路径	当前页面 path ，必须是以 / 开头的完整路径

其中 path，是设置分享页面的路径，其就是我们在 app.json 文件的 pages 字段的数组里配置的页面地址，分享出去的就是这个页面，因此自然填写的 path 路径是必须在 app.json 文件的 pages 字段里存在的。

简单通过如下代码即可实现分享：

```
Page({
  //分享内容
  onShareAppMessage: function () {
```

```
    return {
      title: '小程序分享标题',
      desc: '在小程序中实现分享非常方便',
      path: 'view/api-login/api-login'
    }
  }
})
```

8.6 获取二维码

我们可以通过接口：

https://api.weixin.qq.com/cgi-bin/wxaapp/createwxaqrcode?access_token=ACCESS_TOKEN

来获取小程序任意页面的二维码，获取后，用户通过扫一扫就可以进入小程序的相应页面。

其中 access_token 的获取需要了解微信公众号的 access_token 获取方法，access_token 是公众号的全局唯一接口调用凭据，现在来看，它也会成为小程序的全局唯一接口调用凭据，公众号和小程序调用各微信接口时都需使用 access_token，其有效期只有两小时，两小时后需要重新获取，获取后之前的 access_token 会失效，因此开发者需要实现逻辑在服务端保存 access_token，并在调用时检查其时效，失效后需要重新通过接口获取，其实现逻辑需要读者通过开发者文档相关内容来了解并实现。网上其实也有已实现的代码，可供参考。

该接口需要通过 POST 方式请求，其 POST 参数如表 8.19 所示。

表 8.19 接口参数说明表

	参 数	默认值	说 明
1	path		不能为空，最大长度 128 字节
2	width	430	二维码的宽度

其中参数说明如下。

（1）path，也是我们在 app.json 文件的 pages 字段的数组里配置的页面地址，因此填写的 path 路径是必须在 app.json 文件的 pages 字段里存在的。

（2）width，即生成的二维码的宽度。

举一个简单的 POST 请求例子：

```
{
"path": "pages/index?query=1",
"width": 430
}
```

另外关于二维码，官方做了一些说明，需要读者注意：

（1）通过该接口，仅能生成已发布的小程序的二维码。

（2）可以在开发者工具预览时生成开发版的带参二维码。

（3）带参二维码只有 100000 个，请谨慎调用。

8.7　手把手教你做 Demo——简易登录页

本章所述内容的主要内容涉及较为复杂的服务端实现，在实际开发中，通常这一部分工作都由后端工程师完成相应功能的开发，为前端提供相应的 API 来调用，当然也可以由前端工程师通过 node.js 技术来实现服务端逻辑，但这依然涉及到了较为复杂的服务端技术，这一节，我们先实现一个简单的登录页，并实现页面分享。

为了便于开发与理解，我们将项目目录以最简化组织为如图 8.7 所示。

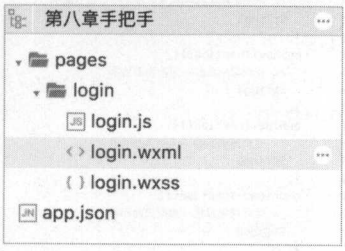


图 8.7

其中 app.json 文件的代码编写为：

```
{
  "pages": [
    "pages/login/login"
  ],
  "window": {
    "navigationBarTitleText": "8-Demo"
```

```
    }
}
```

接下来我们打开 login.js 文件，并在编辑器里输入代码 page，则编辑器会显示代码提示，如图 8.8 所示。

图 8.8

从中选择方框所示的 Page 模板，则代码编辑器会自动补全小程序生命周期函数，如图 8.9 所示。

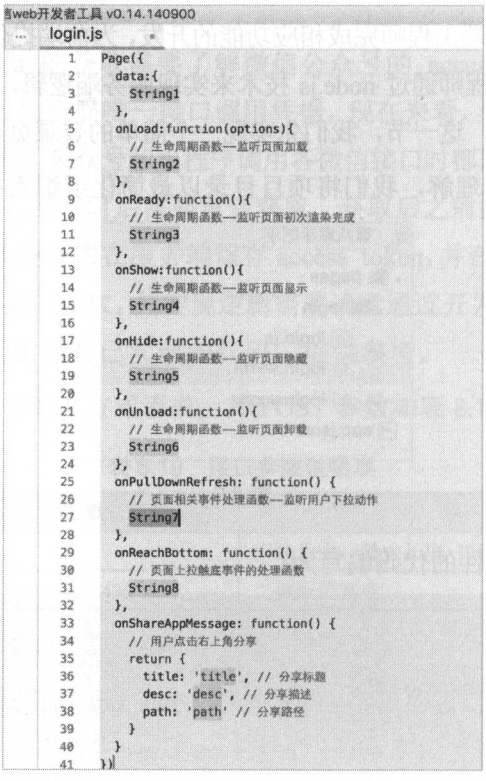


图 8.9

我们只保留需要的 data 对象，以及 onReady 和 onShareAppMessage 方法，为了便于代码结构清晰，我们先删除其他代码，实际开发中，建议保留其生命周期函数及注释。

完成后 login.js 文件的完整代码则为：

```
Page({
  data:{
     "avatarUrl":"",
     "nickName":""
  },
  onReady:function(){
    // 生命周期函数--监听页面初次渲染完成
    var _this=this;
    wx.login({
      success: function(res) {
        if (res.code) {
          console.log('用户登录成功');
            wx.getUserInfo({
               success: function(res) {
                  var userInfo = res.userInfo
                  var nickName = userInfo.nickName
                  var avatarUrl = userInfo.avatarUrl
                  var gender = userInfo.gender//性别 0: 未知、1: 男、2: 女
                  var province = userInfo.province
                  var city = userInfo.city
                  var country = userInfo.country
                  _this.setData({
                     "avatarUrl":avatarUrl,
                     "nickName":nickName
                  })
                  console.log('成功获取了用户信息: ',res);
               }
            })
        } else {
          console.log('获取用户登录态失败! ' + res.errMsg)
        }
      }
    });
```

```
  },
  onShareAppMessage: function() {
    // 用户点击右上角分享
    return {
      title: '第 8 章手把手', // 分享标题
      desc: '第 8 章手把手分享描述', // 分享描述
      path: 'pages/login/login' // 分享路径
    }
  }
})
```

代码中，onShareAppMessage 方法既注册了该页面的分享，其中唯一需要注意的是，path 属性的路径是与 app.json 文件里的 pages 属性的指定路径是一致的。其他属性非常容易理解，title 既分享页面的标题，desc 为分享页面的描述。

在 login.wxml 文件中编写如下代码：

```
<view class="main">
    <view><image src="{{avatarUrl}}" mode="aspectFit"></image></view>
    <view><text class="userinfo-nickname">{{nickName}}</text></view>
</view>
```

并在 login.wxss 文件中编写如下简单的样式代码：

```
.main{
    text-align: center;
    margin-top:30rpx;
}
```

保存以上代码，即可在模拟器中看到如图 8.10 所示的运行效果。

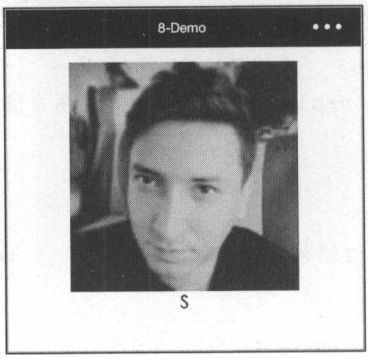


图 8.10

用简单的结构和代码，既获得了用户的头像和昵称等基本信息。在实际开发中会稍稍复杂一些，但总体思路基本类似。

并且也完成了对分享页的注册，将该页面分享到群或者好友的效果如图 8.11 所示。

图 8.11

8.8　本章要点总结

本章将微信官方介绍的开放接口部分梳理了一遍，以简化大家的理解。

但是本章的内容，大部分需要服务端的配合来完成，这一部分工作，在实际工作当中，通常需要与后端开发人员配合来完成，当然前端人员自行通过 node.js 来实现也未尝不可，这就要求前端人员在 node.js 方面有一定的开发经验，由于关于服务端的实现超出了本书的内容，这里就不对后端部分的实例展开来讲，但是本章已经较为清晰地介绍了具体的开发思路，对这一块感兴趣的读者，应该是可以通过学习本章的内容很快了解小程序中使用微信开放接口的开发思想，并迅速开始着手来实现。相信针对这一部分的内容，网络中也会出现非常多的现有资源来使用。

但无论是否准备自行编写后端实现，了解本章内容也是很有必要的，通过了解本章内容，可以加深读者通过小程序调用微信开放接口的方法和思想，对开发出具有完善功能的小程序来说有重要意义。

策划编辑：刘伟

合作伙伴：
轻课（www.qingclass.com）

网络销售：
JD（www.jd.com）
当当（www.dangdang.com）
亚马逊（www.z.cn）
博文视点（www.broadview.com.cn）

微视角，洞察变革先机！

微信公众号：i6v1206

随时为你服务
QQ：17269702
豆瓣：@myheartflies
E-mail：6v1206@gmail.com

微信高级
产品经理
喻佼焱

野狗实时
通信云CEO
刘之

猿圈创始人
郑萌

轻课CEO
肖逸群

乐视网前端
技术副总监
李国建

朝夕日历CEO
程昊

轻课

博文视点Broadview

新浪微博
weibo.com
@博文视点Broadview

策划编辑：刘　伟
封面设计：王　乐

上架建议：程序设计

ISBN 978-7-121-31331-8

定价：59.80元